

# Delivering a product that "just works"

## 

## Hiring Lab - Solutions Engineer - M Mraihi - April 27, 2018

## Step 1 : Setup the environment

In order to prepare the set up for the lab, I set up on my Mac a vagrant environment with an Ubuntu Linux distribution.

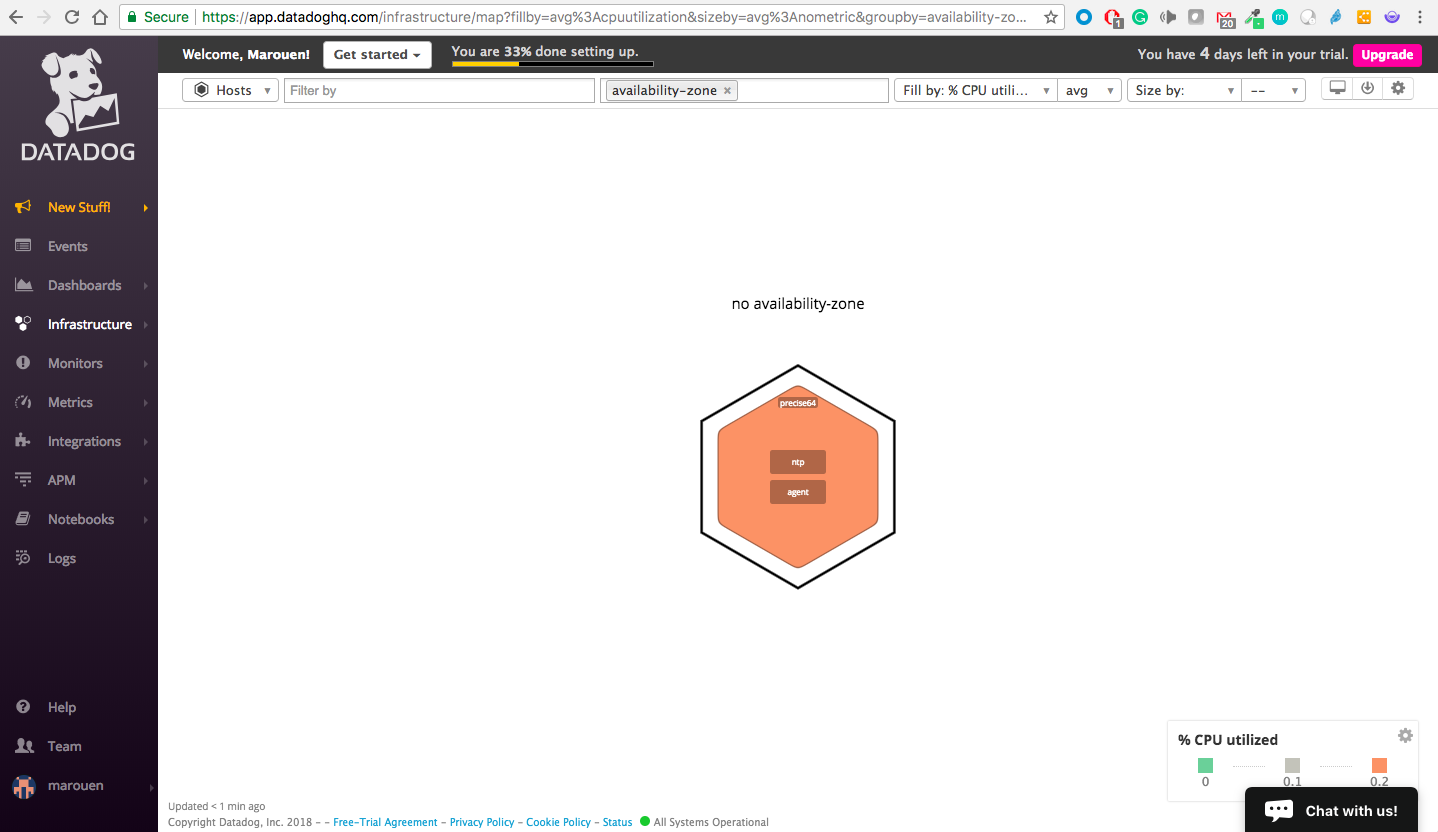
In the process, I created an evaluation account in the Datadog website and followed the assistant to install the agent in my VM with this documentation

<https://app.datadoghq.com/account/settings#agent/ubuntu>

<https://docs.datadoghq.com/agent/basic_agent_usage/ubuntu/>

Few things had to be installed before installing the agent. Agent V6 was deployed.

Once the agent is installed, the assistant moved to the next page showing the infrastructure view as you see here

****

## 

## Step 2 : Collecting Metrics

Of course, the collected data is raw and can be aggregated only using its source. If more reporting dimensions are needed we need to add Tags to these feeds. Using tags enables us to observe aggregate performance across a number of hosts and narrow the set further based on specific elements.

Following the documentation in this link <https://docs.datadoghq.com/getting_started/tagging/assigning_tags/> I added the requested tags in the agent configuration file as below

**Adding Tags**

*Editing the config file*

cd /etc/datadog-agent/

sudo vi datadog.yaml

*Updated :*

# Set the host's tags (optional)

# tags:

# - mytag

# - env:prod

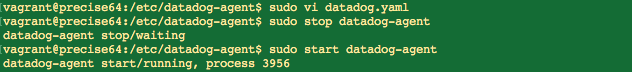
# - role:database

tags:

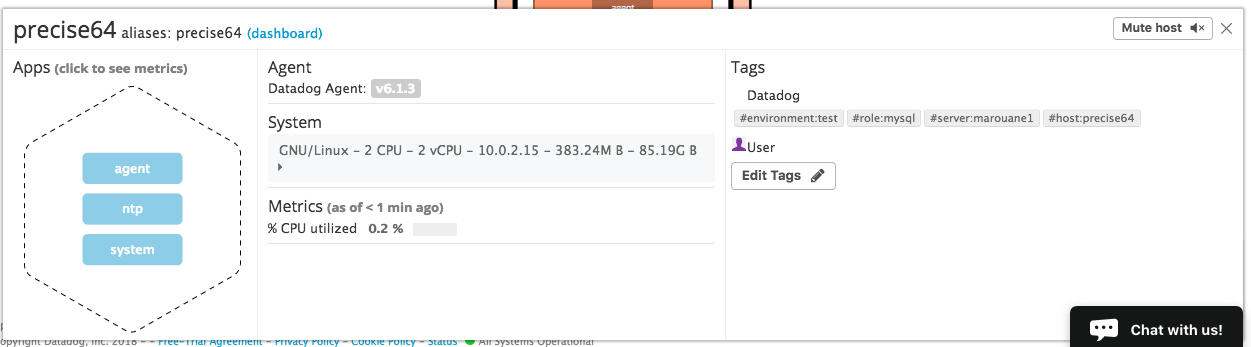
- server:Marouane1

- environment:test

- role:mysql



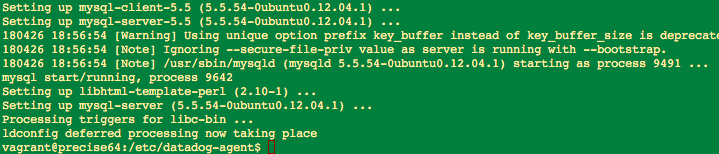
The results were visible in the hostmap after restarting the agent. Now we can trace back the information to the tags we added.

Results in the host map : 

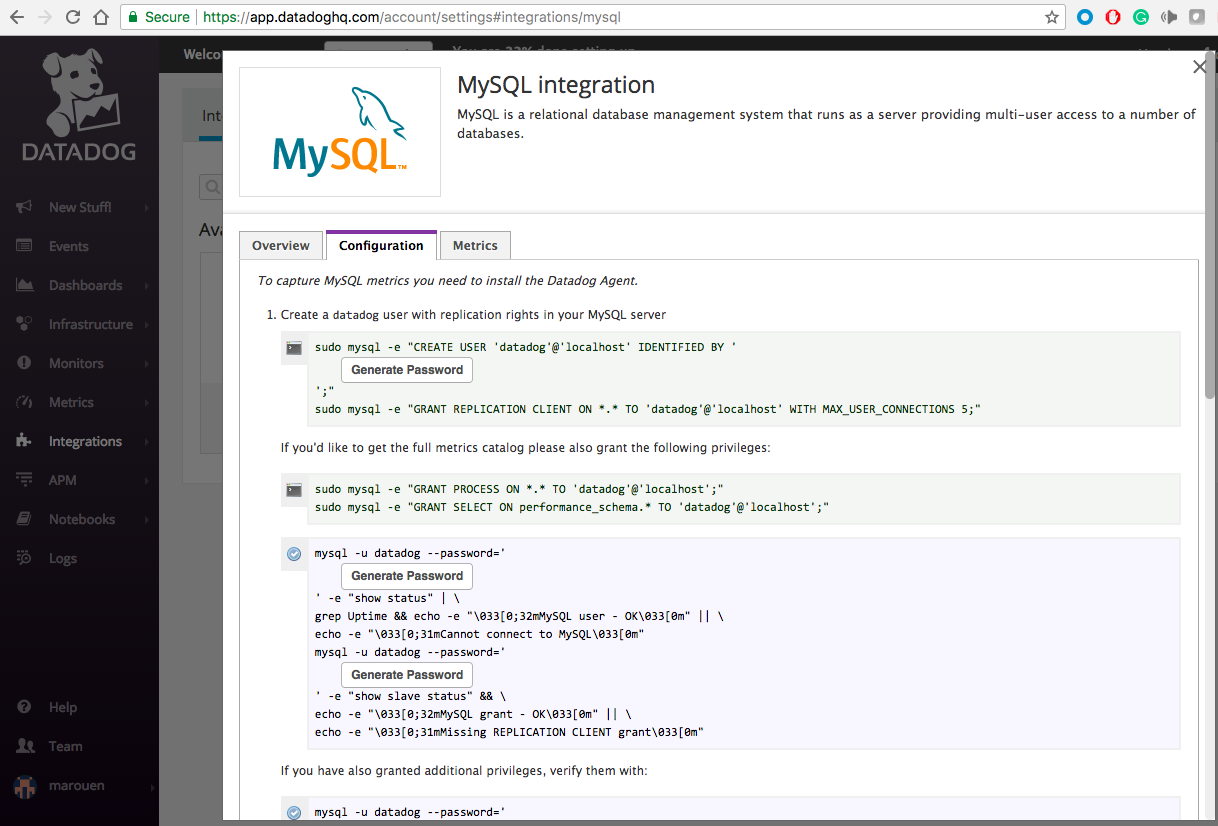
Datadog can collect more than infrastructure data. For instance, we are going to add here an integration with a MySQL server installed in the same VM to collect DB performance metrics and show them into our Datadog dashboards.

**Installing Mysql database**

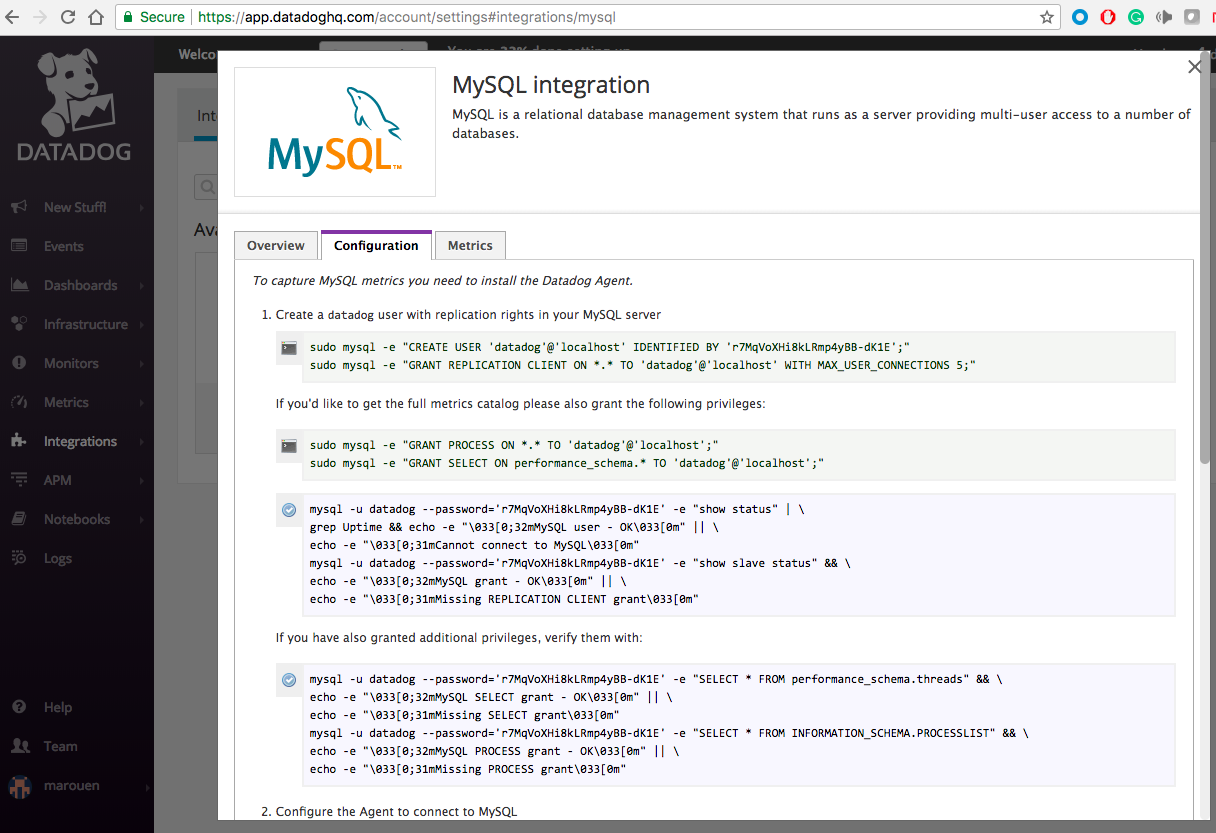
sudo apt-get install mysql-server



Adding the mysql integration using the assistant/documentation in this link <https://app.datadoghq.com/account/settings#integrations/mysql>



Generating the random password



*The password need to be provided with the parameter --password*

*Commands used*

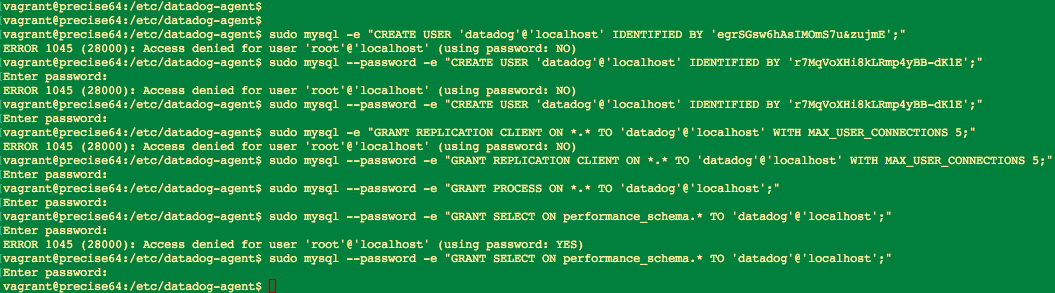
sudo mysql --password -e "CREATE USER 'datadog'@'localhost' IDENTIFIED BY 'r7MqVoXHi8kLRmp4yBB-dK1E';"

sudo mysql --password -e "GRANT REPLICATION CLIENT ON \*.\* TO 'datadog'@'localhost' WITH MAX\_USER\_CONNECTIONS 5;"

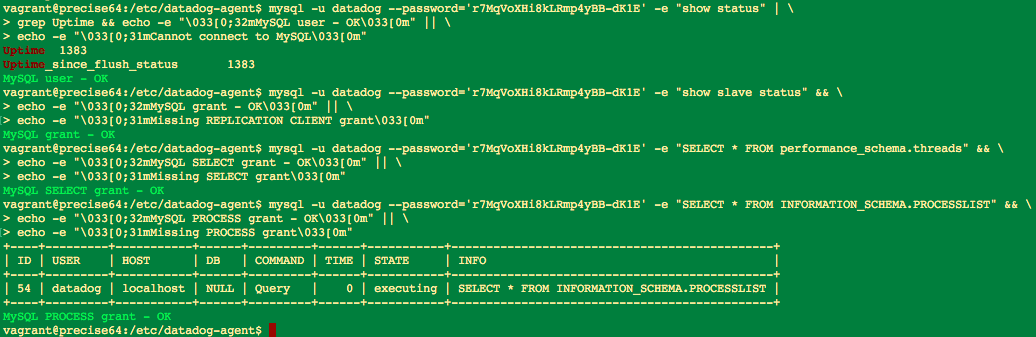
sudo mysql --password -e "GRANT PROCESS ON \*.\* TO 'datadog'@'localhost';"

sudo mysql --password -e "GRANT SELECT ON performance\_schema.\* TO 'datadog'@'localhost';"

Results



Testing the rights



Configuring the integration config file

*Adding the lines*

- server: localhost

user: datadog

pass: r7MqVoXHi8kLRmp4yBB-dK1E

tags:

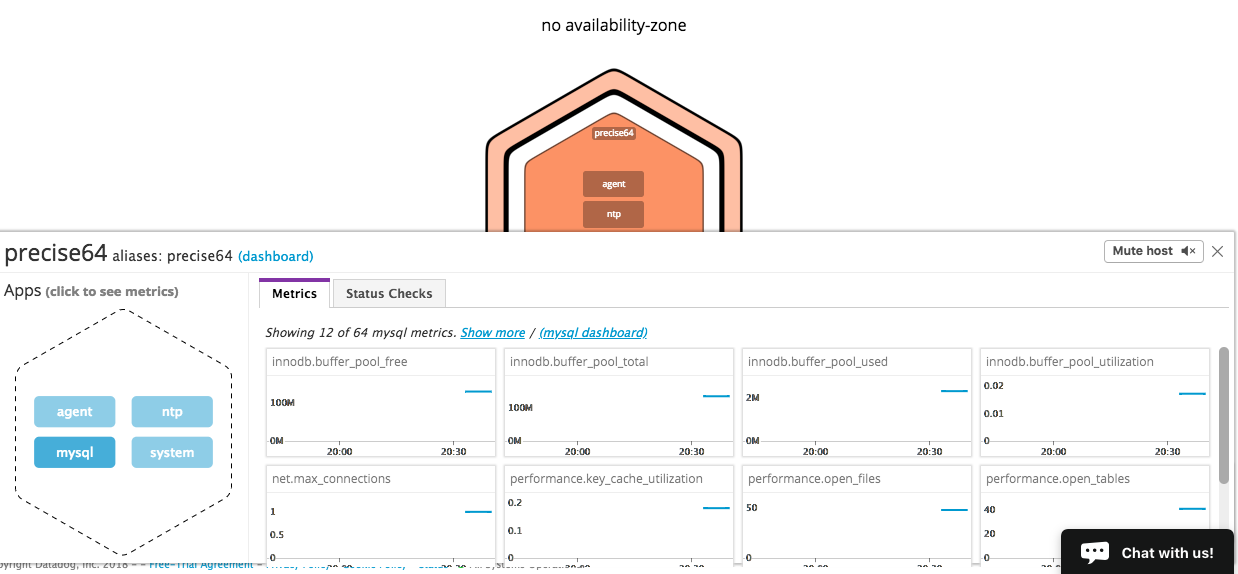
options:

replication: 0

galera\_cluster: 1

Restarting the agent

The hostmap confirms that the integration was successful and we started receiveing the data from MySQL through our agent



Sometimes, we will need to focus on custom metrics that are not provided out of the box (ie. for a bespoke application or for a specific metrics not provided by the OS). For this it is possible to add a custom agent.

**Creating a custom agent check**

Based on the documentation : <https://docs.datadoghq.com/agent/agent_checks/>

*Creating the metric config file :* conf.d/my\_metric.yaml

*Content:*

init\_config:

instances:

[{}]

*Creating the script : checks.d/my\_metric.py*

import random

from checks import AgentCheck

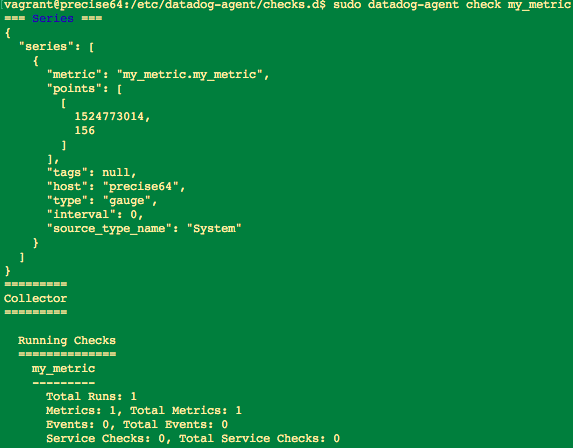
class HelloCheck(AgentCheck):

def check(self, instance):

self.gauge('my\_metric.my\_metric', random.randint(0, 1000))

Testing the script with the command line : sudo datadog-agent check my\_metric

Results :



To change the interval to 45 seconds (the default is 15 seconds), the config file conf.d/my\_metric.yaml has been updated to

init\_config:

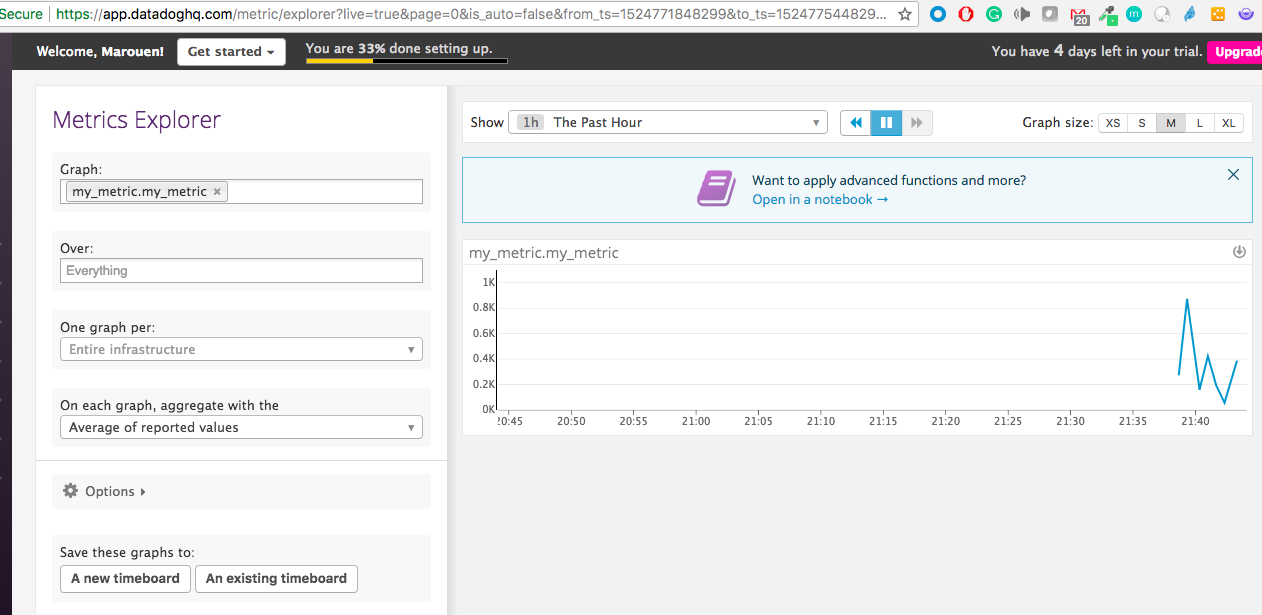
instances:

[{

min\_collection\_interval: 45

}]

After restarting the agent, the dashboard will show



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

**Answer :** I guess that I already used the parameter in the YAML file to have the interval changed to 45. If I had to repeat the lab and use the script instead, I think I can pause the script with the function time.sleep() with 30 seconds : time.sleep(30)

## 

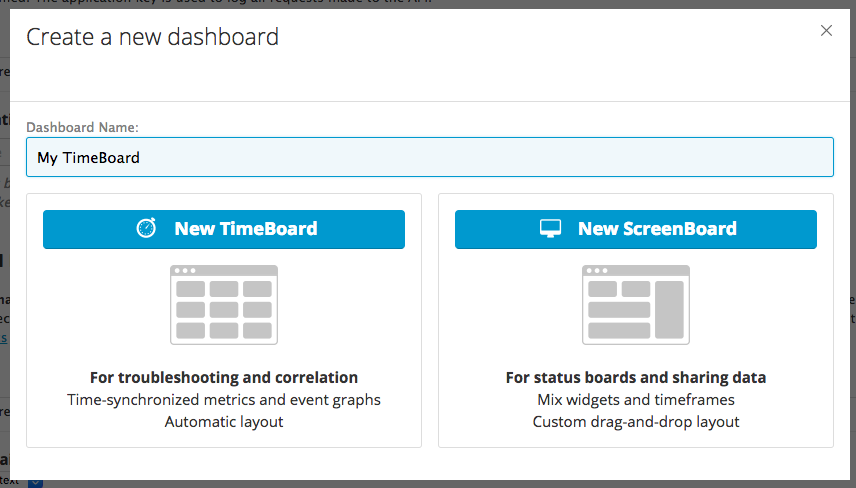
## Part 3 - Visualizing Data

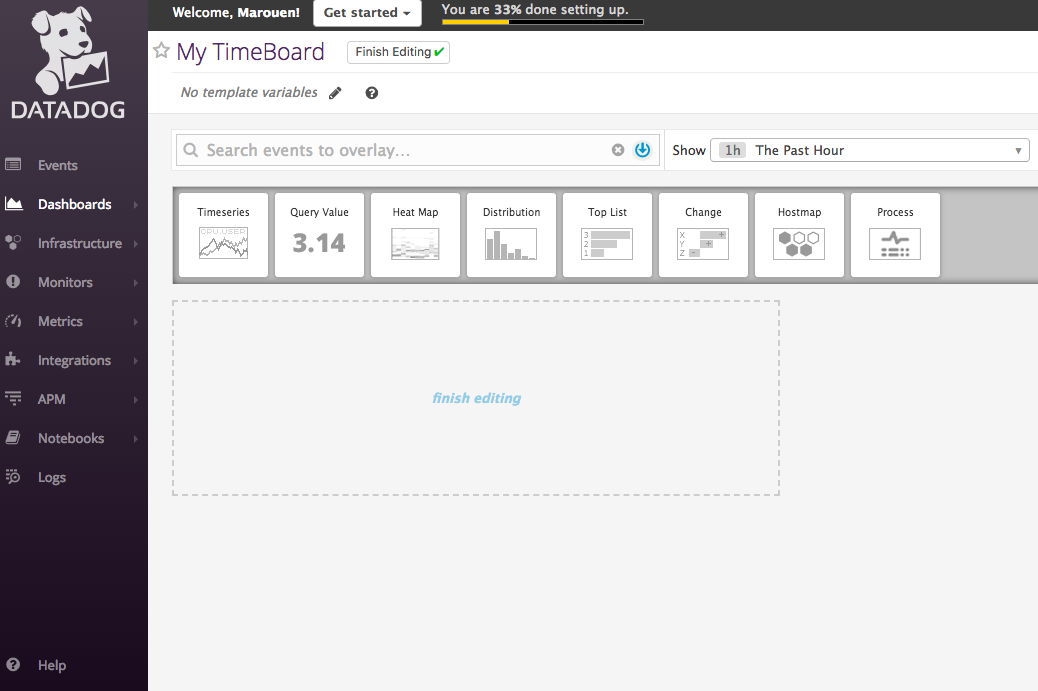
With DataDog, it is easy to create Dashboards and add the metrics and monitors from the data collected. This can be done very easily using the GUI. Datadog allows also to create these dashboards using APIs from outside.

To show this, I will start by creating a dashboard using the GUI just to collect the JSON script to use it later from outside via a REST call.

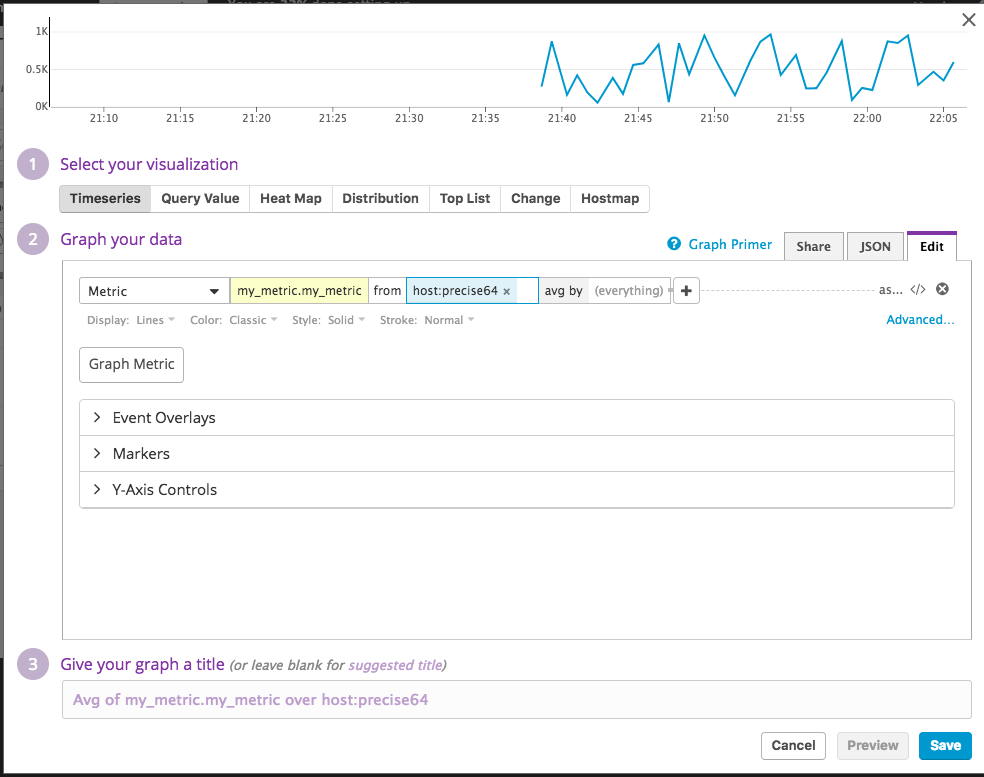
**Creating the Timeboard**

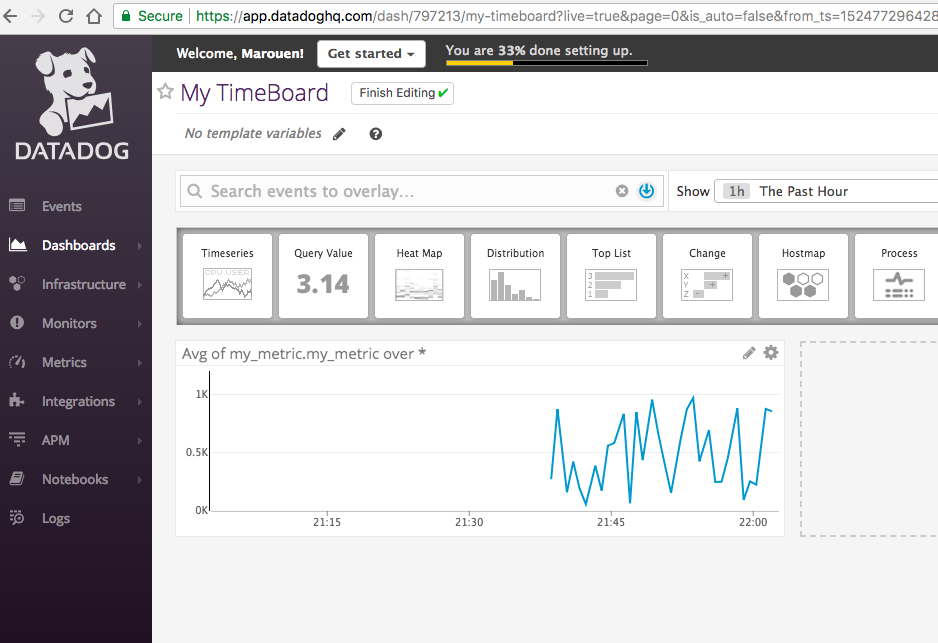
I started by selecting New Dashboard from the GUI. In our case, we need a TimeBoard to start with.



I started with a timeseries graphic. 

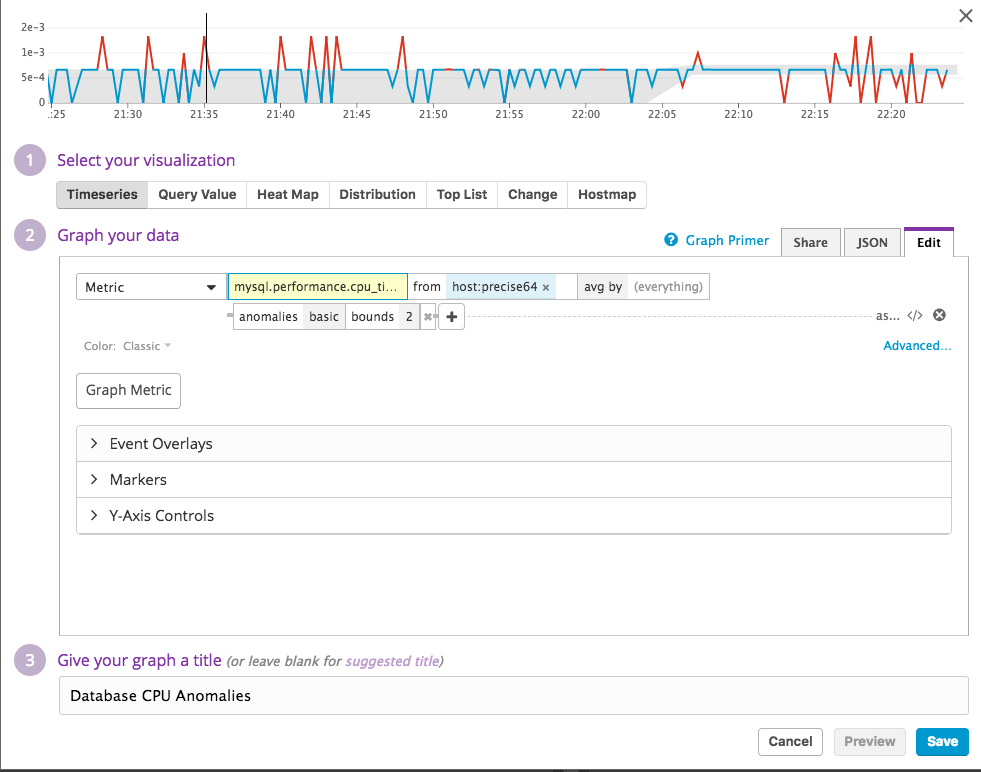
I selected after the custom metric I created earlier with the conditions defined in the exercice. 





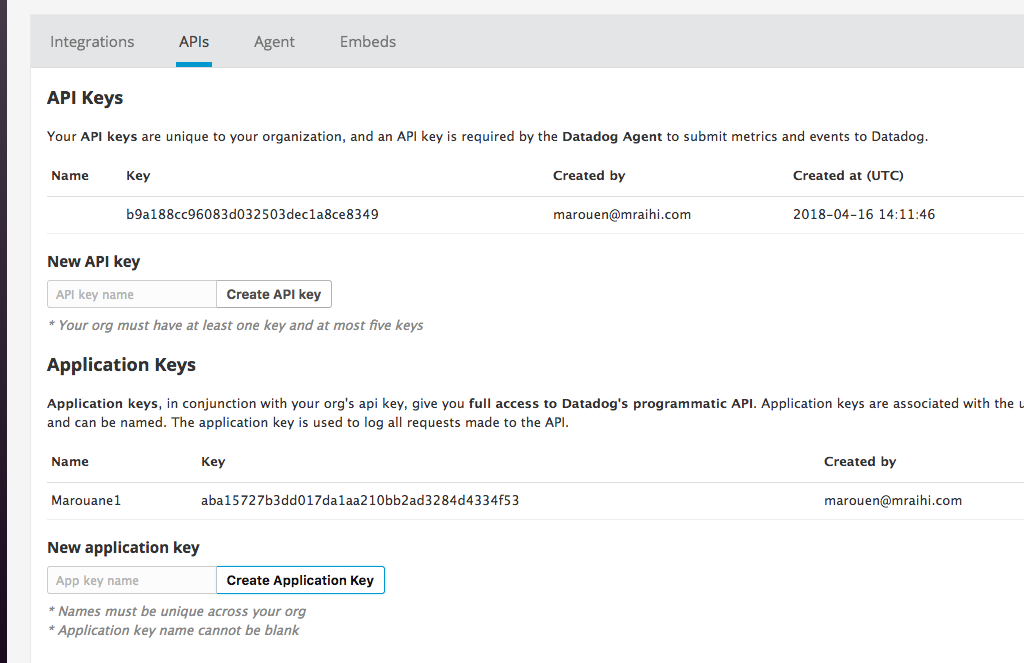
Once the Graphic is added, I added another graphic for an anomaly using this documentation <https://docs.datadoghq.com/monitors/monitor_types/anomaly/>

For this, I created another timeseries focusing on the Database performance with the anomaly option as shown below



Selecting the JSON tab, this will show the JSON code used to create this graphic in case it needs updating manually. This code will be copied and used in the REST API call.

Because I will use an API call, I will need to get the authentication details (API KPI and Application KPI) that is needed for the call.

Creating the Application API security Key to be used with the Account API Key

Using a timeboard via API using this documentation <https://docs.datadoghq.com/api/?lang=python#create-a-timeboard>

After extracting the JSON scripts from the timeseries created, the code is included in this API call

{

"graphs": [{

"title": "My Metric Graph",

"definition": {

"events": [],

"requests": [{

"q": "avg:my\_metric.my\_metric{host:precise64}.rollup(sum,3600)"

}]

},

"viz": "timeseries"

},

{

"title": "MySQL CPU Time Anomalies",

"definition": {

"events": [],

"requests": [{

"q": "anomalies(avg:mysql.performance.cpu\_time{host:precise64}, 'basic', 2)",

"type": "line",

"style": {

"palette": "dog\_classic",

"type": "solid",

"width": "normal"

},

"conditional\_formats": []

}]

},

"viz": "timeseries"

}

],

"title": "Marouane API Timeboard",

"description": "Marouane API Timeboard",

"template\_variables": [{

"name": "host1",

"prefix": "host",

"default": "host:precise64"

}],

"read\_only": "True"

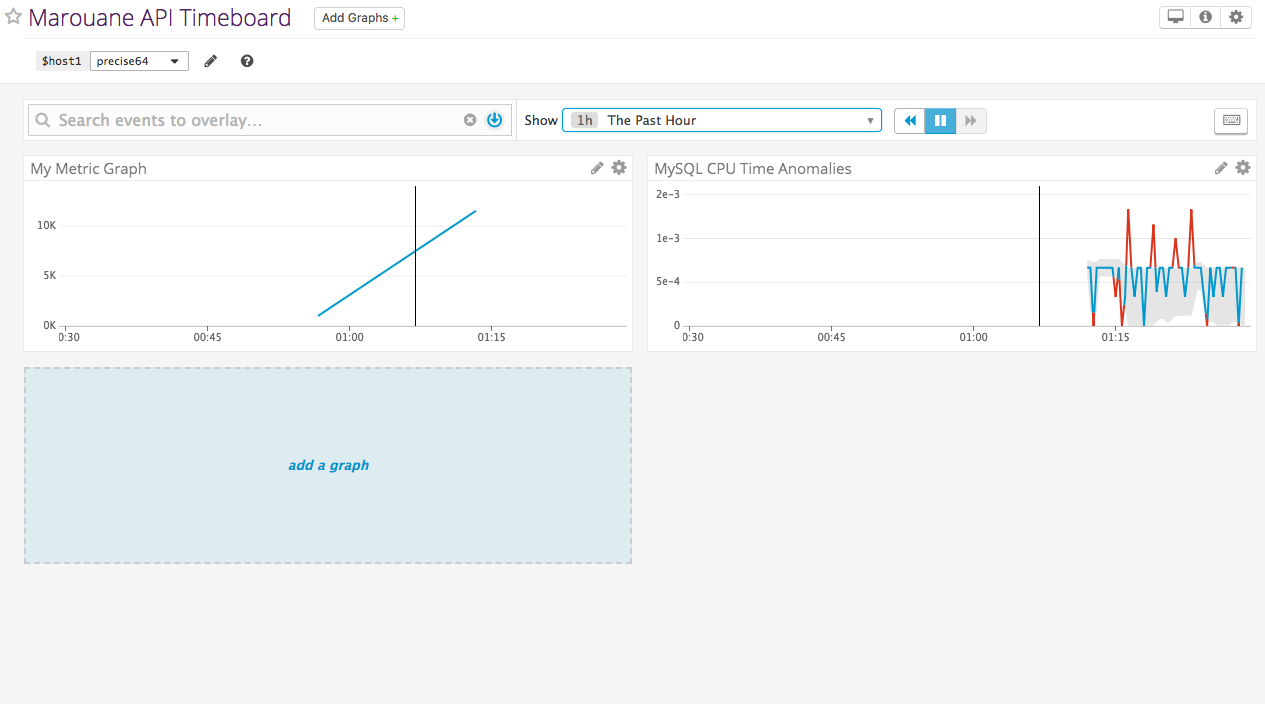
}

Then make a test POST call in Postman to this URL <https://api.datadoghq.com/api/v1/dash?api_key=b9a188cc96083d032503dec1a8ce8349&application_key=aba15727b3dd017da1aa210bb2ad3284d4334f53>

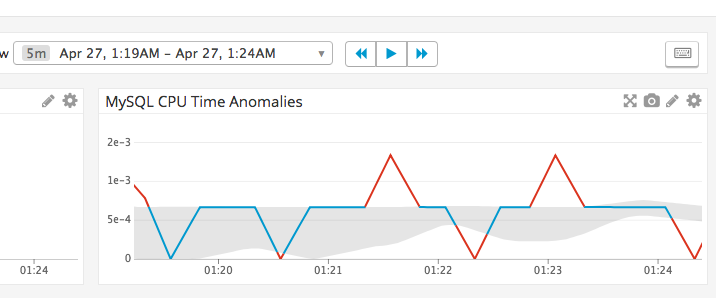


I changed the rollup sum to 1000 seconds to be able to see some results.

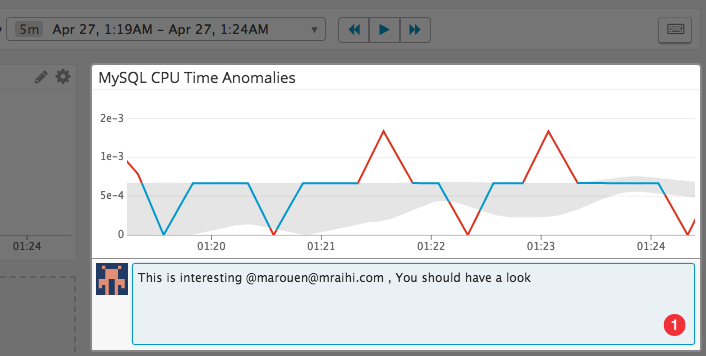
The result is like this



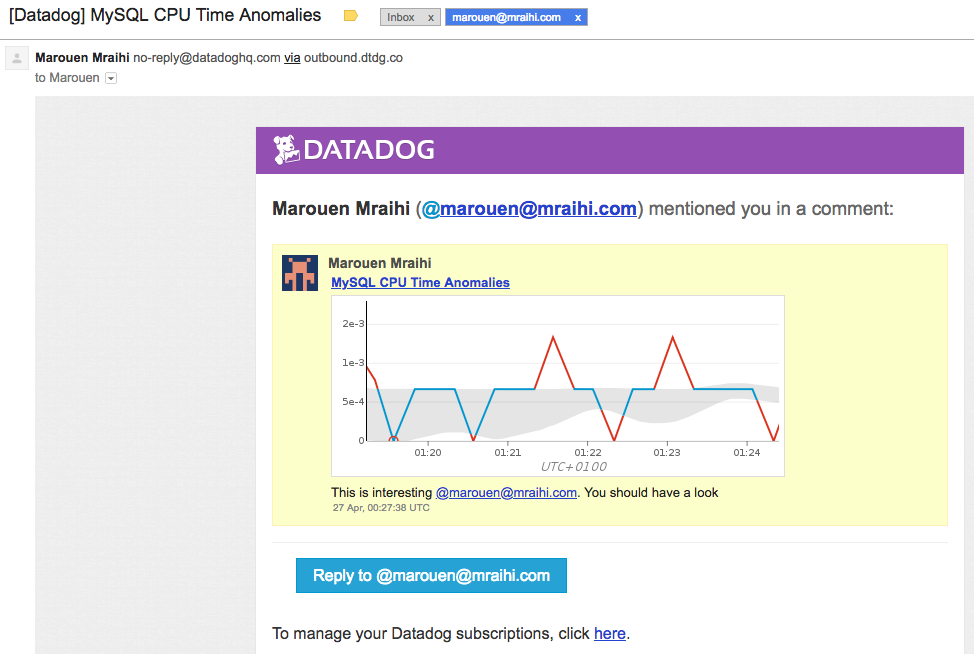
It is possible to restrict the time range to 5 minute by clicking within the graph starting from the end.



Once within the selected range, it is possible to take a snapshot of the results and send it to someone with the platform by mentioning their name after typing @ as you can see below.



This will send an email to this person with the snapshot as below



A reply will start a discussion between the persons tagged like this



* **Bonus Question:** What is the Anomaly graph displaying?
* **Answer :** The anomaly graph displays the real time results in relation with the historically collected results. Historical results are displayed in the shape of a GRAY band with real time data is displayed in BLUE when within the historical band and Red when is considered as an anomaly outside of the usual results.

## 

## Step 4 : Monitoring Data

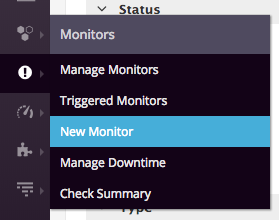
Monitoring all of your infrastructure in one place wouldn’t be complete without the ability to know when critical changes are occurring. Datadog gives you the ability to create monitors that actively check metrics, integration availability, network endpoints, and more.

For this exercise, I will create a monitor that will focus on the custom metric I created earlier and will alert me only in specific cases.

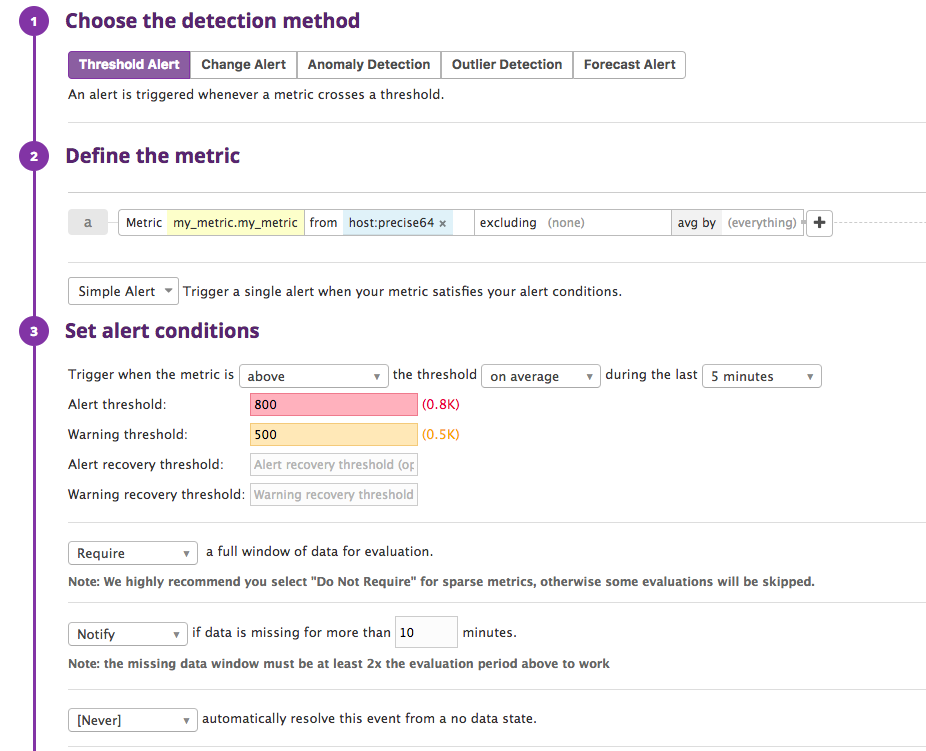
**Creating a new monitor**

Using the documentation <https://docs.datadoghq.com/monitors/monitor_types/metric/>

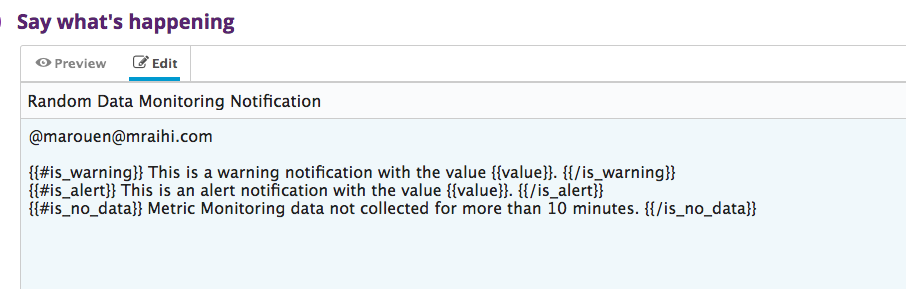
To create a new metric monitor, I selected from the menu **New Monitor** , then **Metric**.



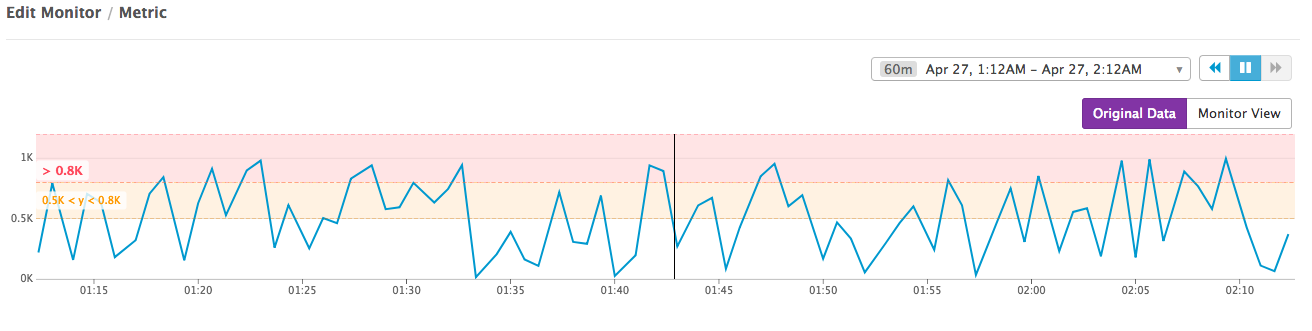
The monitoring was configured as below



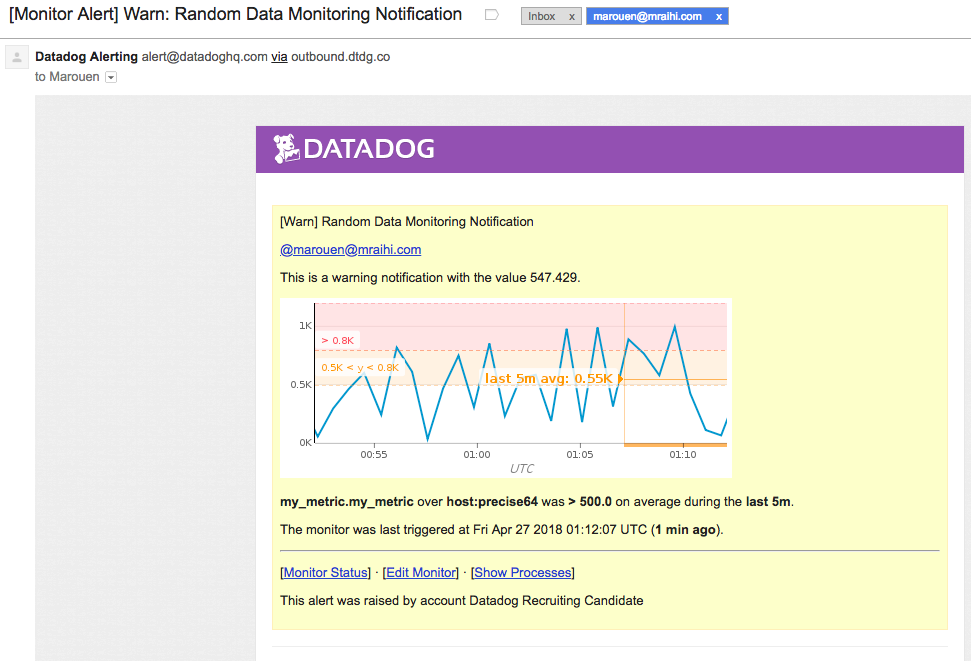
With the content of the notification as below



The configuration page shows also the past of these alerts in the top of the page with the selected thresholds

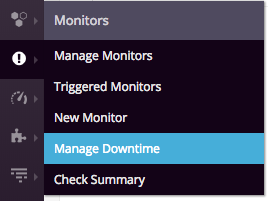


The alert notification received via email is

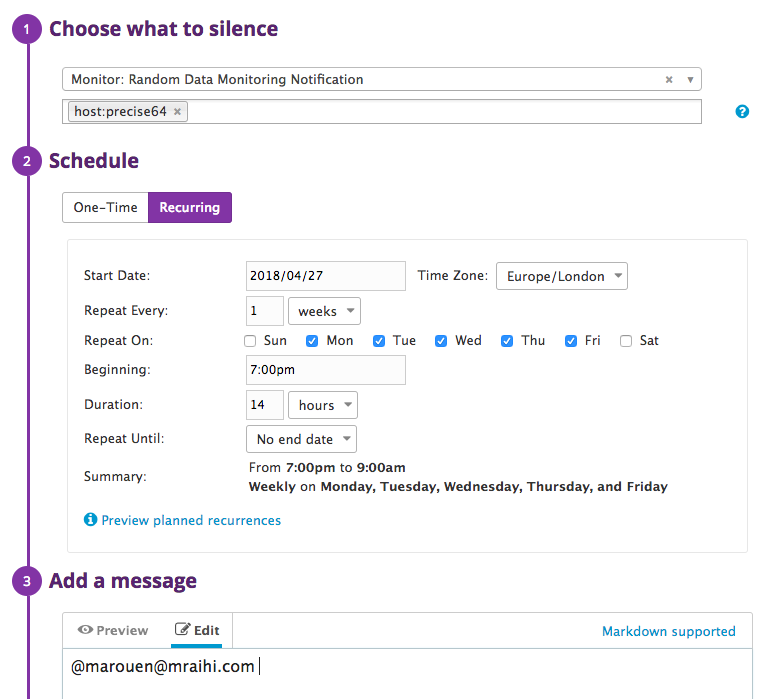


**Bonus Question:** Since this monitor is going to alert pretty often, you don’t want to be alerted when you are out of the office.

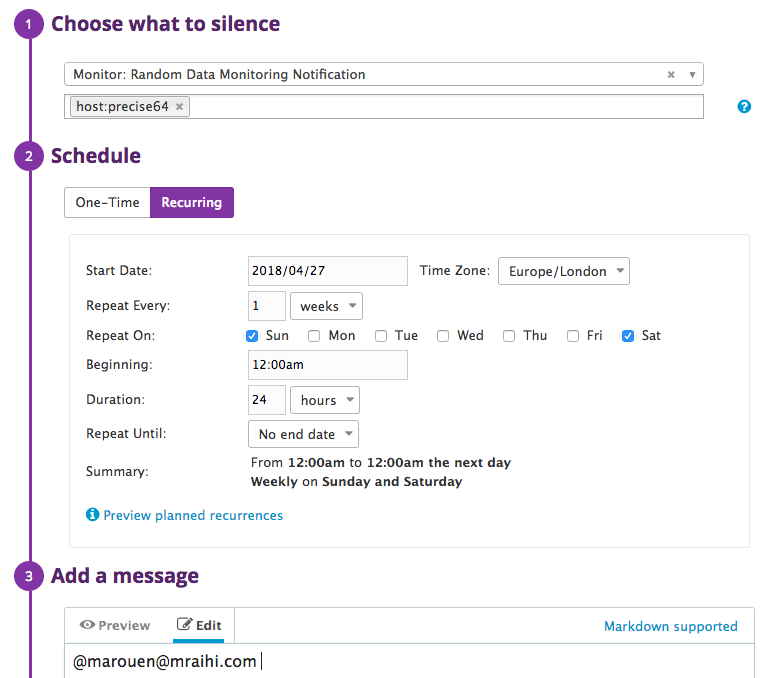
**Answer :** For this we need to use the downtime section as shown here



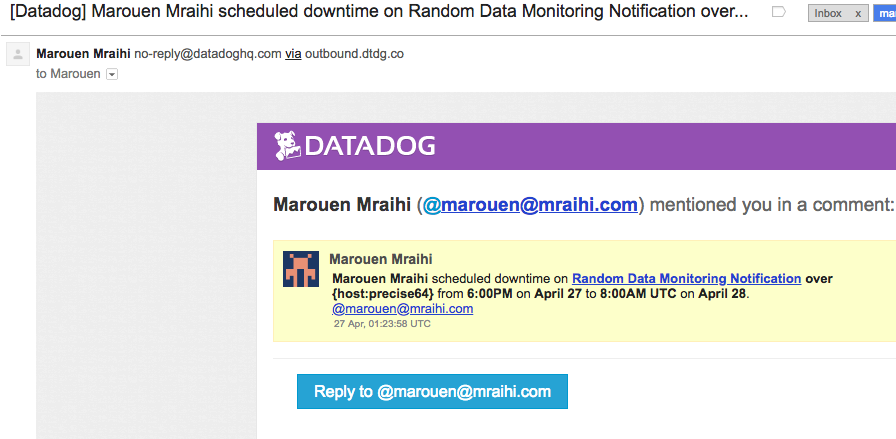
First Downtime : During the week



Second Downtime : Weekend



Notification confirmation



## 

## Step 5 - Collecting APM Data

Datadog APM provides a deep insight into the application’s performance—from automatically generated dashboards monitoring key metrics, such as request volume and latency, to detailed traces of individual requests—side by side with your logs and infrastructure monitoring.

To test this, I was asked to create an web application in the lab VM and monitor it via DataDog APM traces.

I started by looking at this documentation :   
<https://app.datadoghq.com/apm/install>  
<http://pypi.datadoghq.com/trace/docs/#module-ddtrace.contrib.flask>

Preparations : Installing Flask and PIP following this documentation <http://hanzratech.in/2015/01/16/setting-up-flask-in-ubuntu-14-04-in-virtual-environment.html>  
<https://pip.pypa.io/en/stable/installing/>

sudo apt-get update  
sudo apt-get install python-virtualenv  
sudo apt-get install python-pip  
mkdir my\_app  
cd my\_app  
sudo virtualenv env  
source env/bin/activate  
curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"  
sudo python get-pip.py  
sudo apt-get install python-flask

Created application in my\_app.py

from flask import Flask

import logging

import sys

# Have flask use stdout as the logger

main\_logger = logging.getLogger()

main\_logger.setLevel(logging.DEBUG)

c = logging.StreamHandler(sys.stdout)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

c.setFormatter(formatter)

main\_logger.addHandler(c)

app = Flask('myFlaskApp')

@app.route("/")

def hello():

return "Hello World!"

@app.route('/')

def api\_entry():

return 'Entrypoint to the Application'

@app.route('/api/apm')

def apm\_endpoint():

return 'Getting APM Started'

@app.route('/api/trace')

def trace\_endpoint():

return 'Posting Traces'

if \_\_name\_\_ == '\_\_main\_\_':

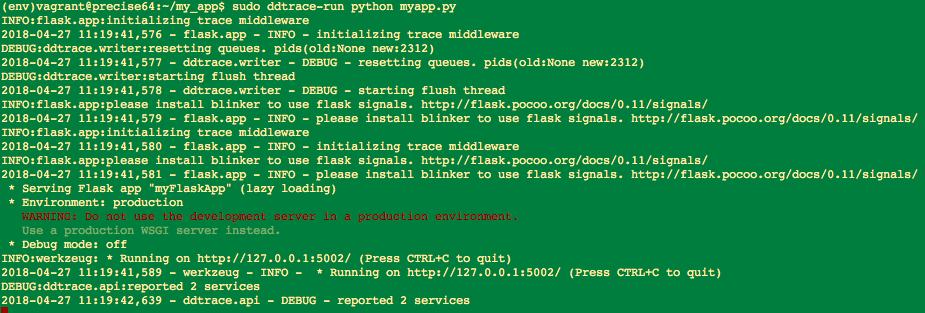
app.run(port=5002)

I did install APM agent ddtrace but I was not able to get any details into DataDog. So I changed the script a bit to make it work

sudo pip install ddtrace

Running the Flask application (I tried ports 5000 and 5001 and I get errors, but port 5002 worked)

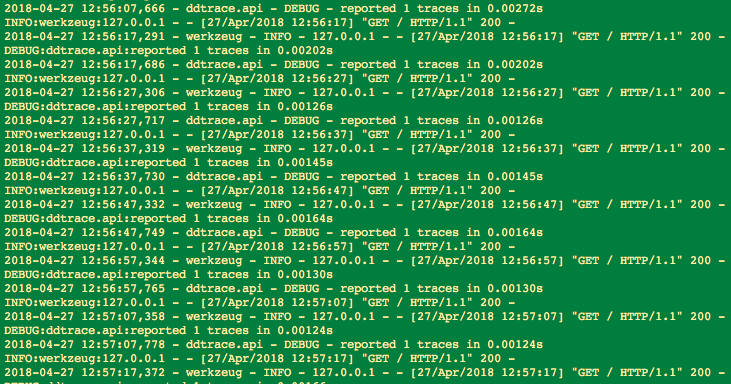
sudo ddtrace-run python myapp.py



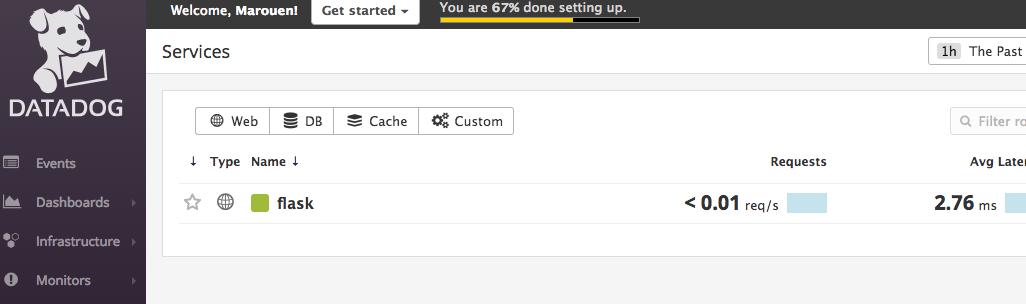
To see the results, I needed to generate some traffic with the command line that will call the webapp every 10 seconds then Datadog will capture these calls (traces)

while true; do curl 127.0.0.1:5002 ; sleep 10; done

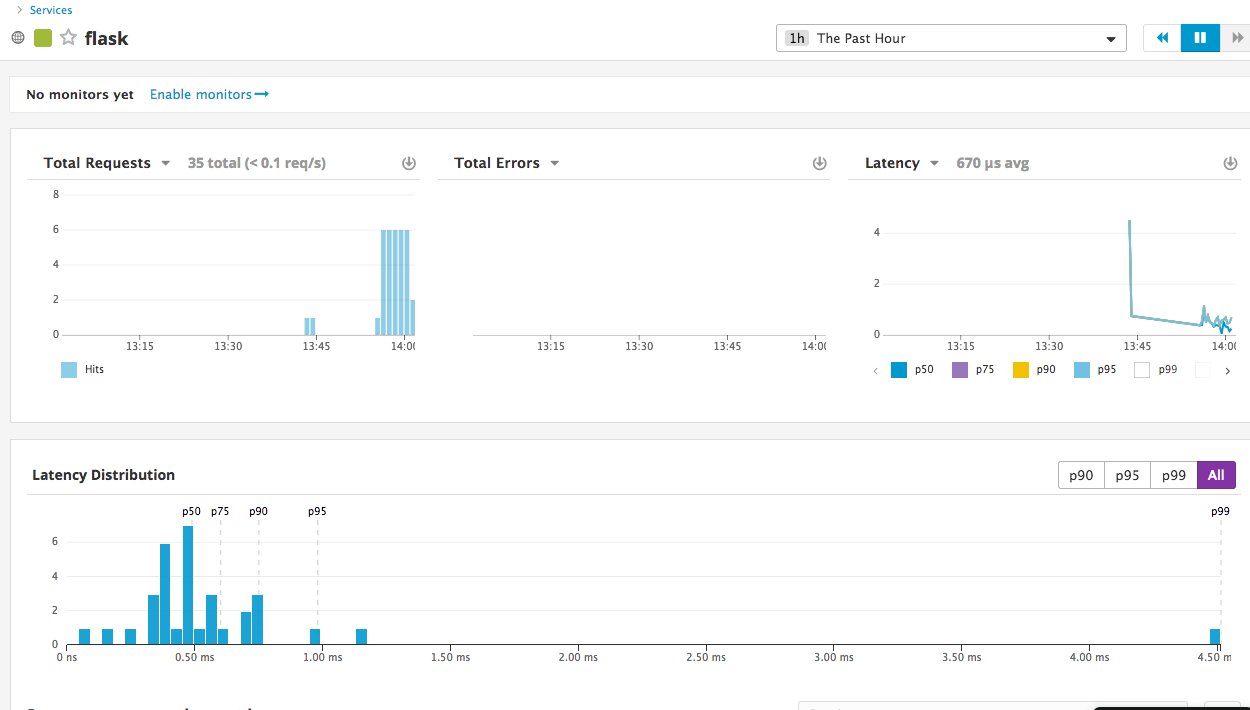
This gives us this result



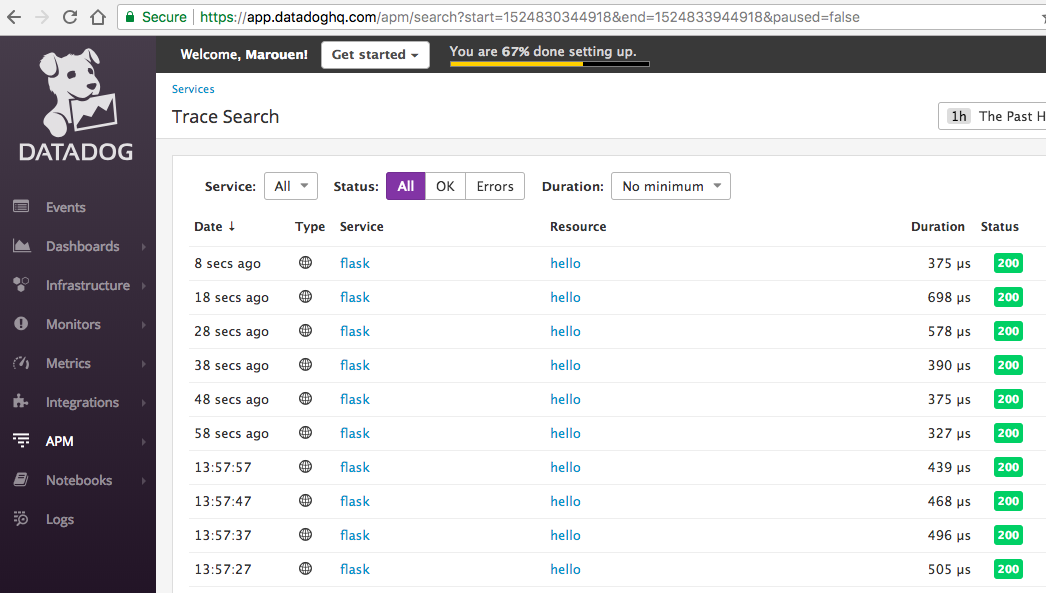
In Datadog GUI, going to the APM section started showing this screen with my application shown as a Service.



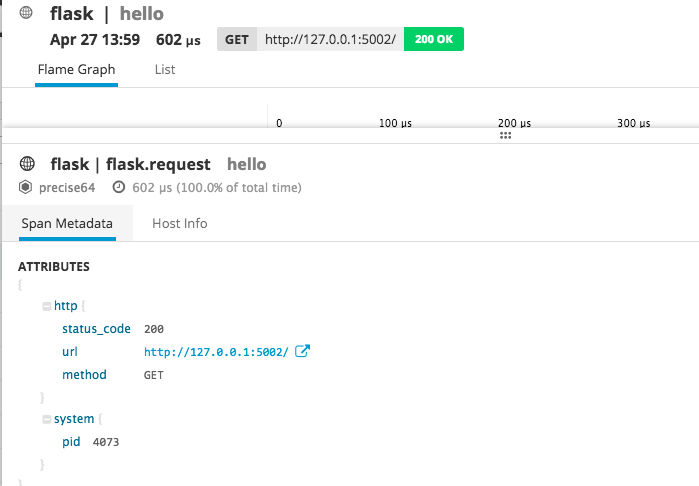
Drilling into the service we get this dashboard



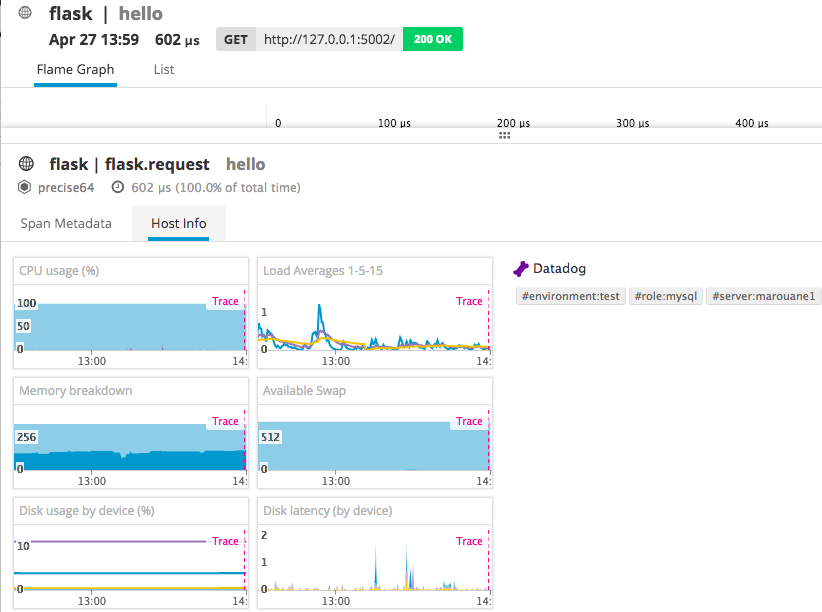
We can also see the details of these traces received by going back to the APM menu and selecting traces.



Selecting any of these traces will give more details like here :



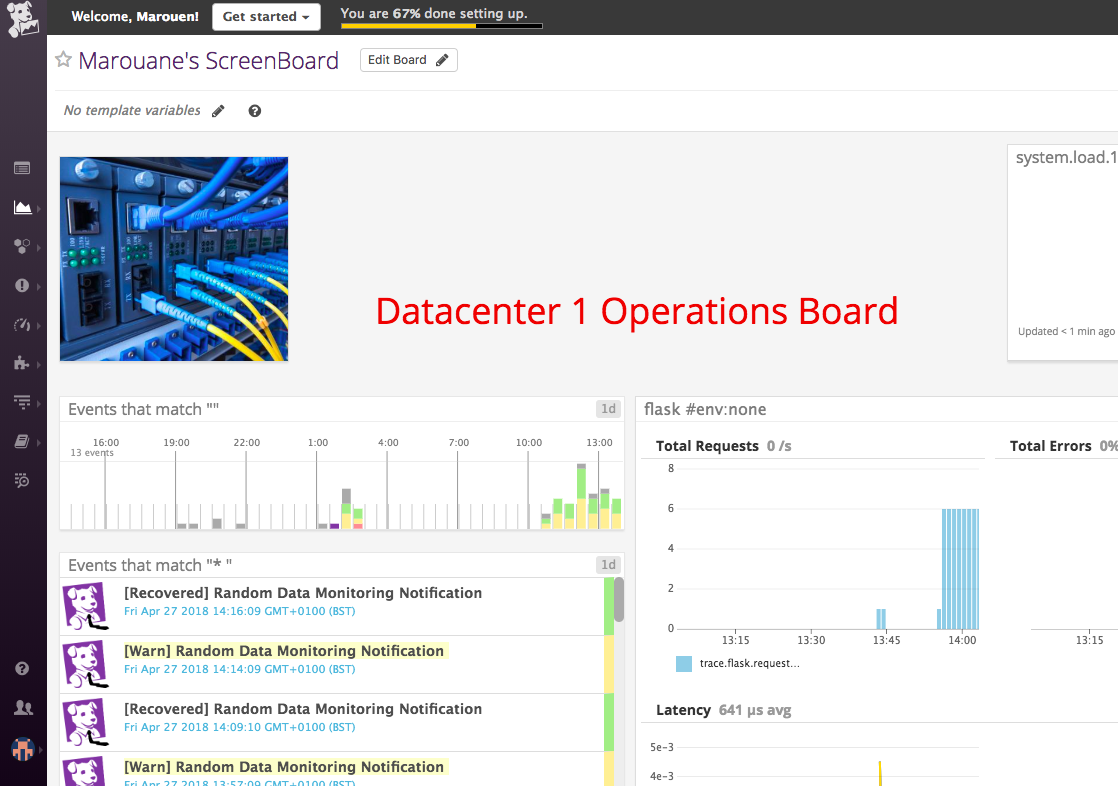
If we change the tab to Host Info, we can even see details about the host performance when that trace/call happened. This helps correlating information in order to investigate a performance issue for example.



As requested, the final step is to create a PUBLIC dashboard including APM and Infrastructure metrics/graphics.

To do that I needed to create a ScreenBoard after clicking ‘New Dashboard’ in the ‘Dashboards’ dropdown.

Using the graphical editor, I added few metrics and graphics to create this view



Then I shared the dashboard using the options in the top right corner

This dashboard is available here

<https://p.datadoghq.com/sb/78120bf87-acf3574ecb72bcc63755ccd24d925d22>

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

**Answer :** In infrastructure and DevOps worlds, a service is a set of processes providing a capability to the user or the department. It can itself also encapsulate other services. A resource is the technical element that helps supporting and providing that service in conjunction with other resources. A resource can be a set of containers supporting a service of an internet banking website.

### 

### **Is there anything creative you would use Datadog for?**

****

It would be maybe interesting to use Datadog with IoT technologies to provide a way of managing the performance of sustainable energy source such as wind turbines. Collecting all the data from the turbines and analysing it in the cloud might help optimizing the performance and maybe even provide feedback for R&D to improve the production and make more innovative changes.