

New Hire Lab Exercise: Response Document

Applicant: Bradley Flemming

February 2018

Contents

[Exercise 1: Collecting Metrics 3](#_Toc506921866)

[Exercise 2: Visualizing Data 9](#_Toc506921867)

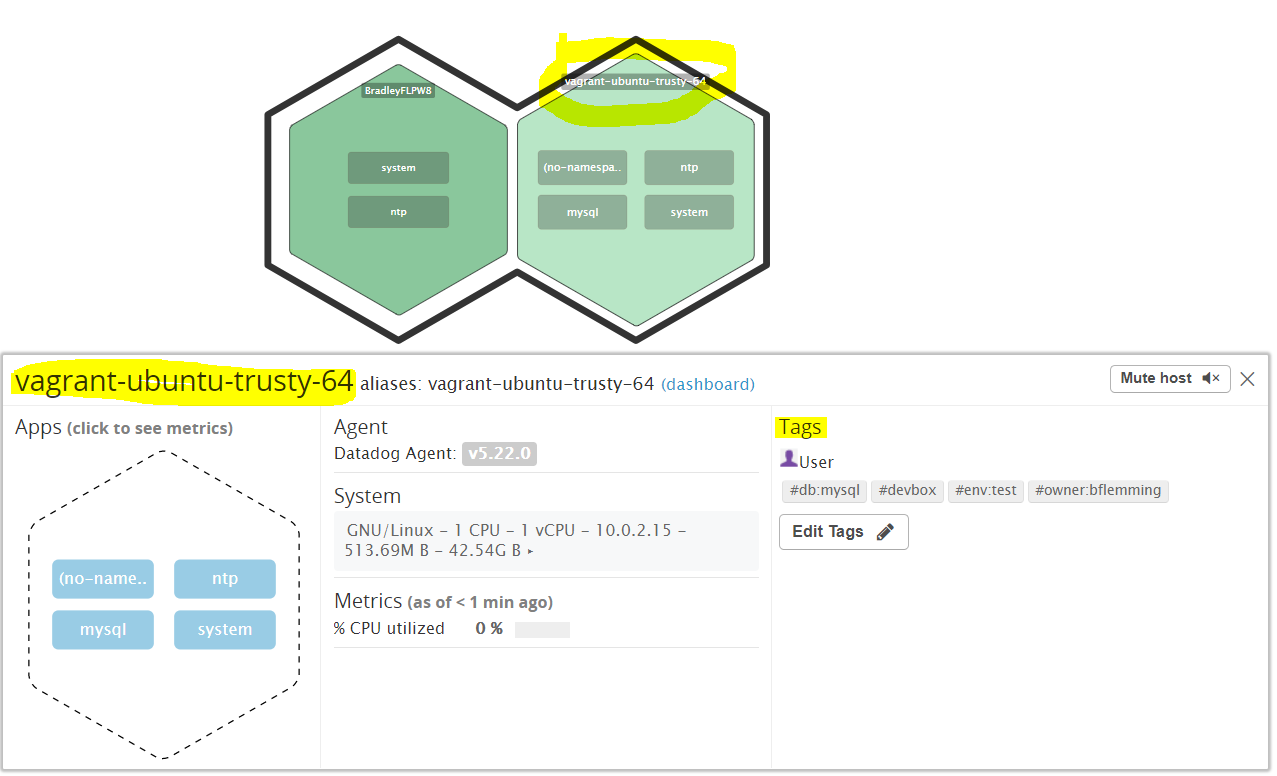
[Exercise 3: Monitoring Data 13](#_Toc506921868)

[Exercise 4: Collecting APM Data 19](#_Toc506921869)

[Final Question – APM Use Case 22](#_Toc506921870)

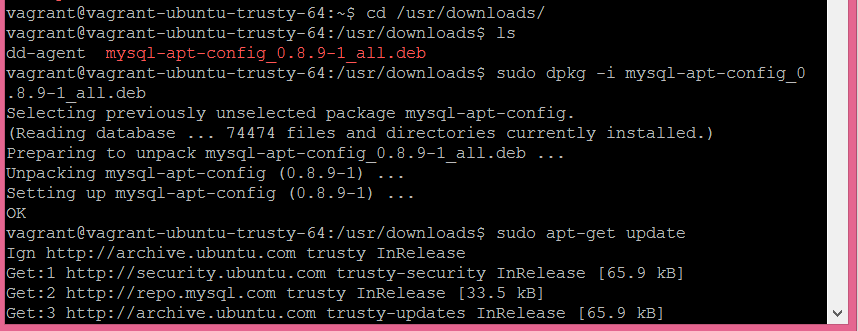
# Exercise 1: Collecting Metrics

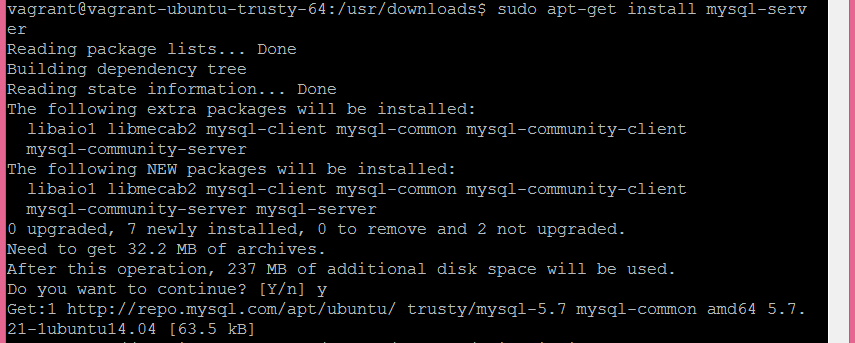
Before we can start collecting metrics we need something to collect metrics from and we need a way to understand where those metrics are coming from and what they mean. Tagging allows us to do that. I applied tags applied to my Ubuntu Linux host by editing the config file (might be a bit small to read):



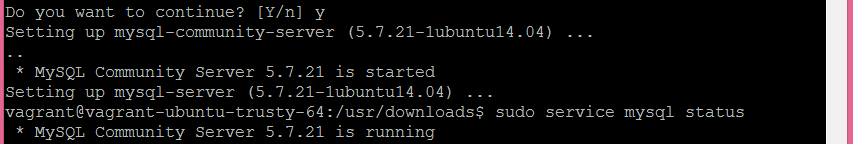
I then installed MySQL on the Ubuntu server so that I could then collect metrics from this application (Simple instructions on how to get MySQL up and running can be accessed here: <https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/#apt-repo-fresh-install> ).

I used the MySQL APT distribution pack to configure and install MySQL (informative Linux install response messages screenshotted below..):





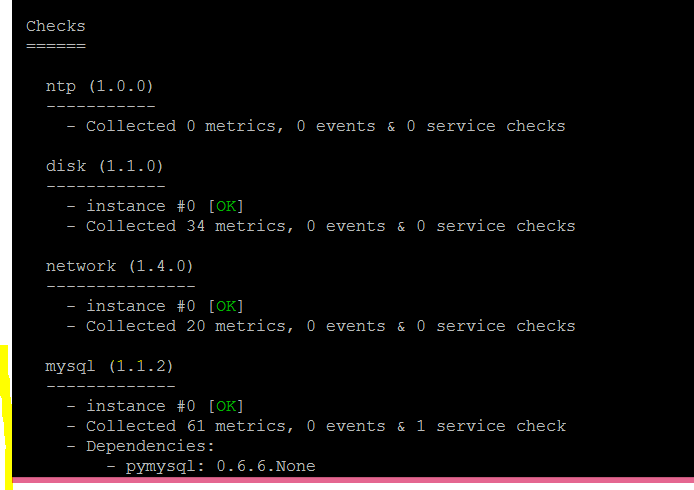
MySQL is running as shown below:



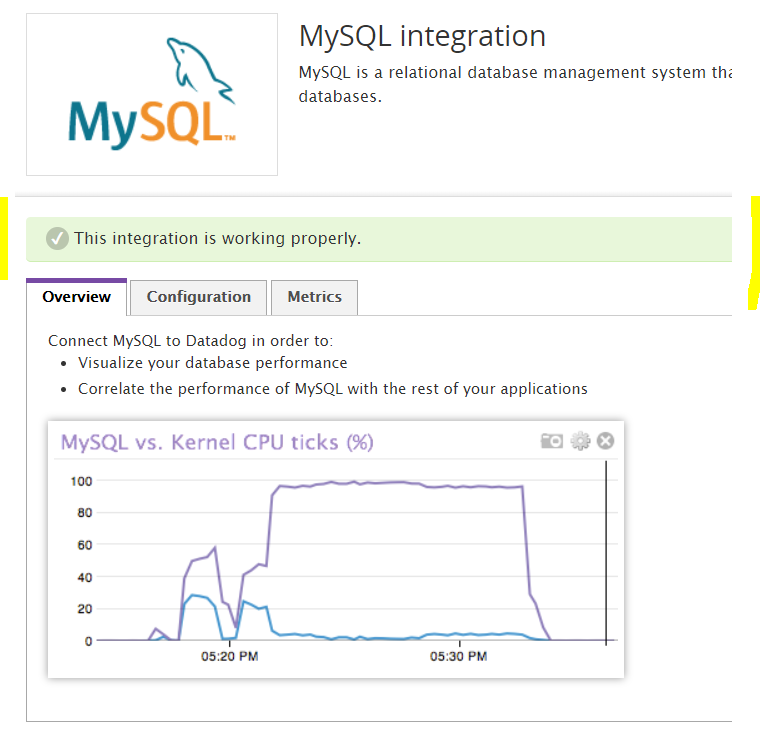
Verified the integration for DataDog was running using:

sudo /etc/init.d/datadog-agent info

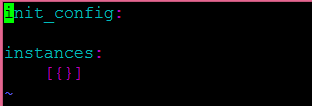
* MySQL Integration applied and running:



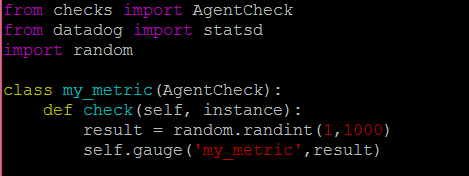
Also verified it was running under the Integrations view on the DataDog UI:



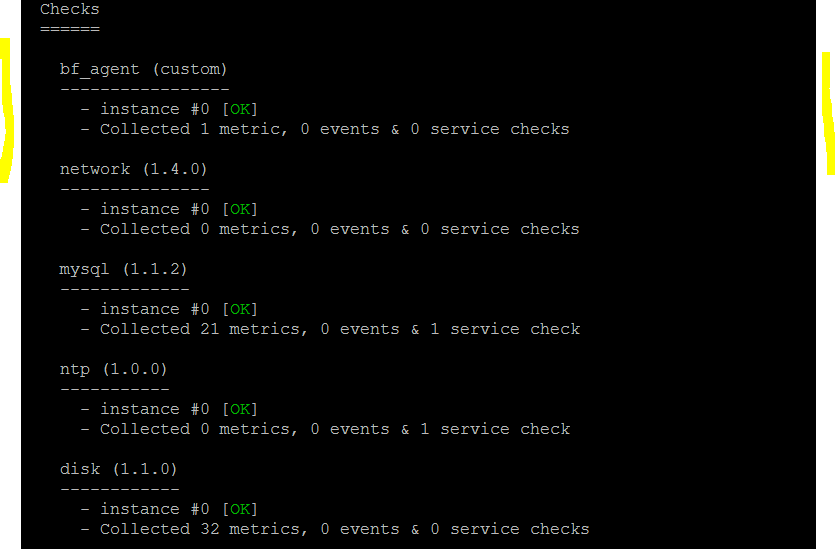
The next step was to create a custom agent called **bf\_agent** which consists of two files, **bf\_agent.yaml** (code below) in /etc/dd-agent/conf.d:



…and **bf\_agent.py** (simple Python code below) in the /etc/dd-agent.checks.d/ folder:

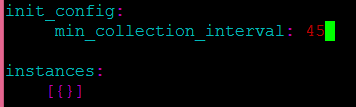


I then invoked the **/etc/init.d/dd-agent** **info** command to verify the new custom agent is running (as seen at the top of the info response below):

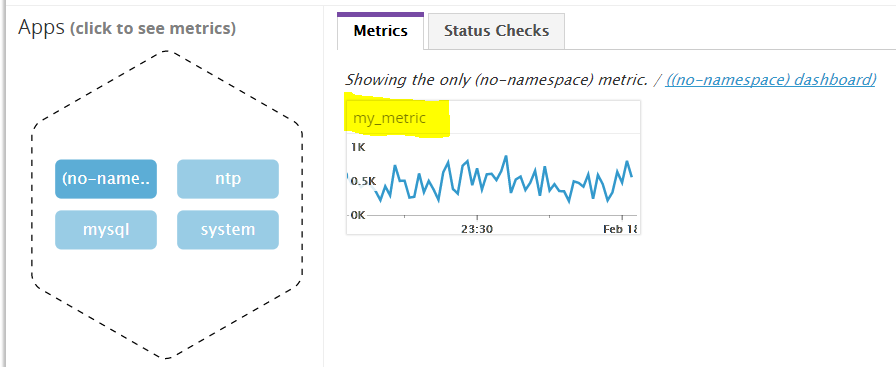


**Bonus Question time!**

I edited the **bf\_agent.yaml** (screenshot below) file to add a collection interval of once every 45 seconds. Altering the YAML file meant I did not have to change the python code:

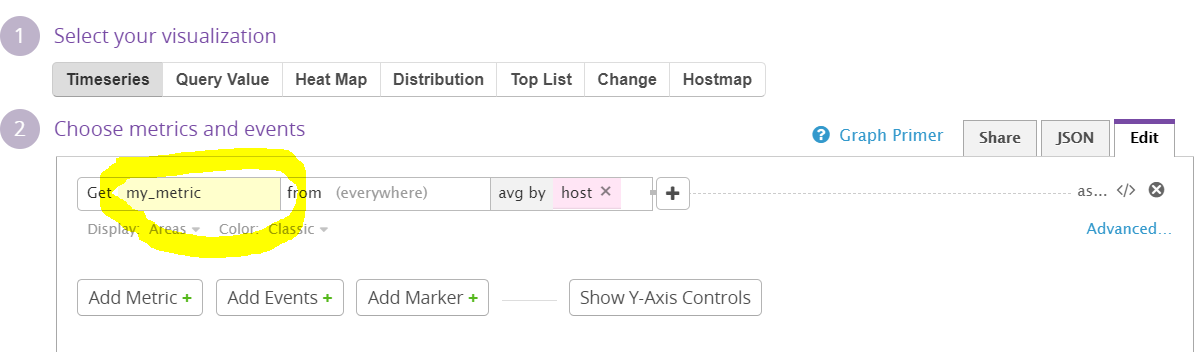


We can now see “my\_metric” under the Host Map in DataDog:

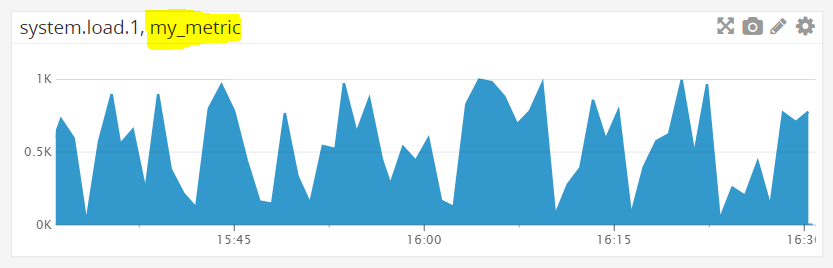


# Exercise 2: Visualizing Data

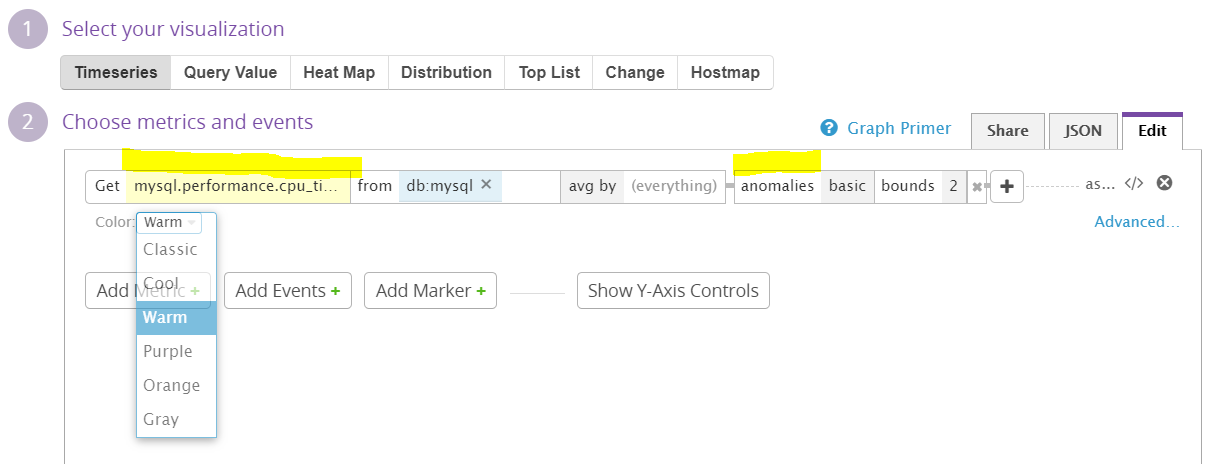
The next task was to have the value of **my\_metric** being visualised on a Timeboard Dashboard. The drop-down list by the ‘Get’ provided a list of all the variables I could use to visualise. For this first graph I selected the “my\_metric” value which gave me a 1-hour view by default as shown below:



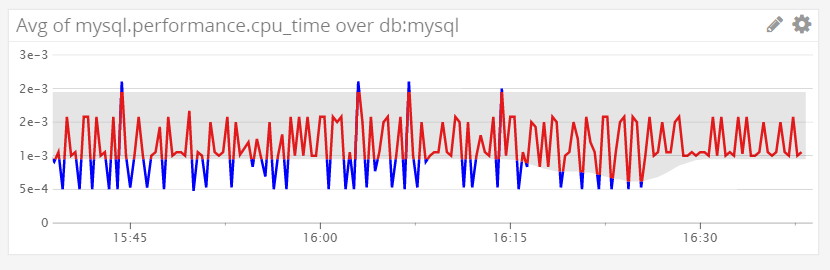
Save and view – time period selected is the past 1 hours activity by default:



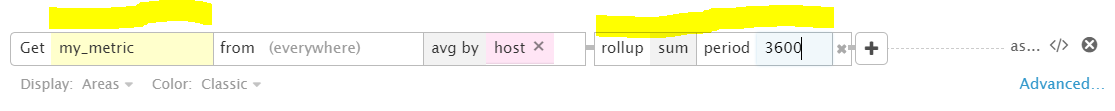
I then dragged a new visualisation widget onto the UI and followed the same process to show metrics for the MySQL application. I chose collecting CPU consumption average by MySQL integration with an anomaly applied (shown below):



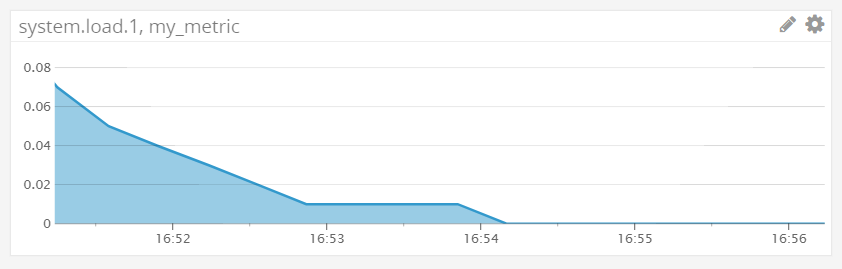
On my Timeboard dashboard this widget looks like:



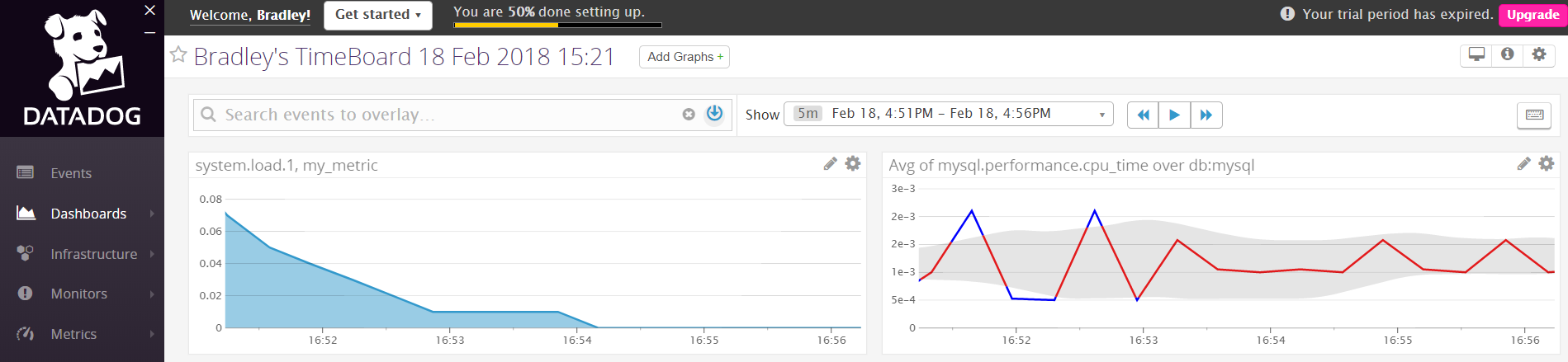
…then to apply the RollUp function to my\_metric and sum over 1 hour (3600 seconds) I added a function by clicking on the ‘**+**’ symbol on the configuration line:



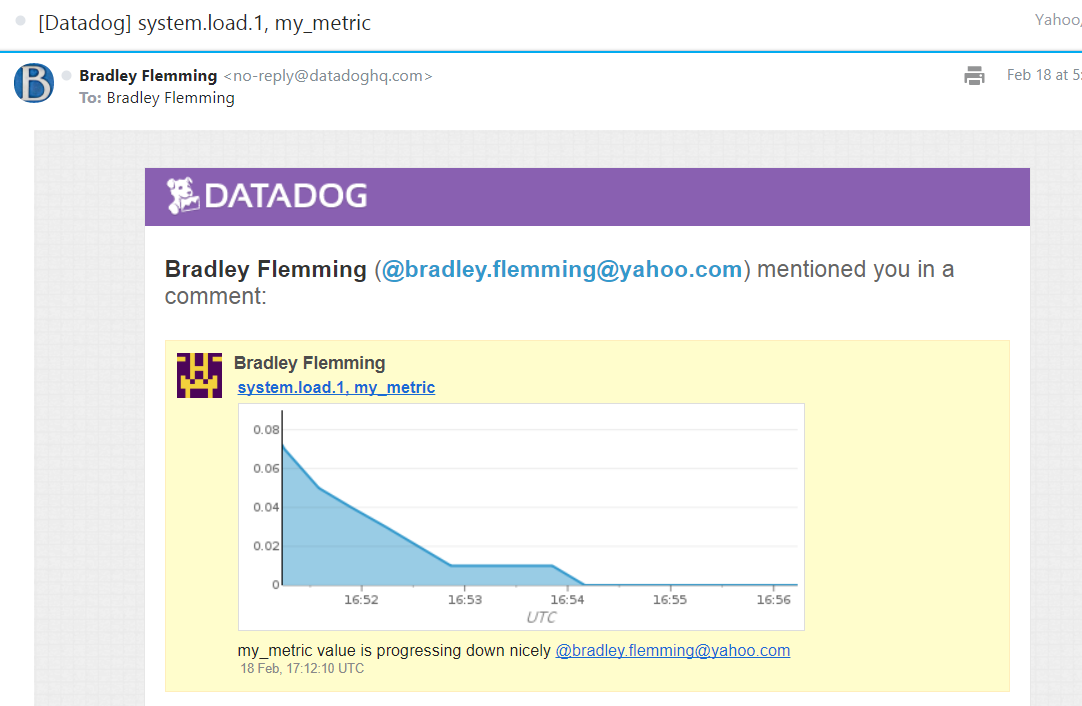
Below is a screenshot of the my\_metric timeline over a 5minute time window:



…and here are the two widgets displayed on the Datadog Dashboard view:



I received a Snapshot notification of this in my e-mail as well showing the my\_metric timeline graph:

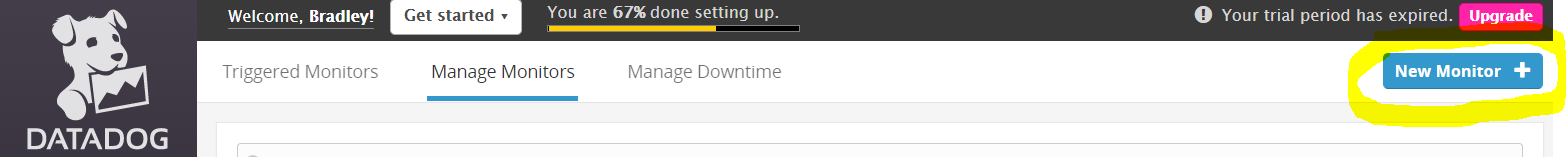


**Bonus Question time!**

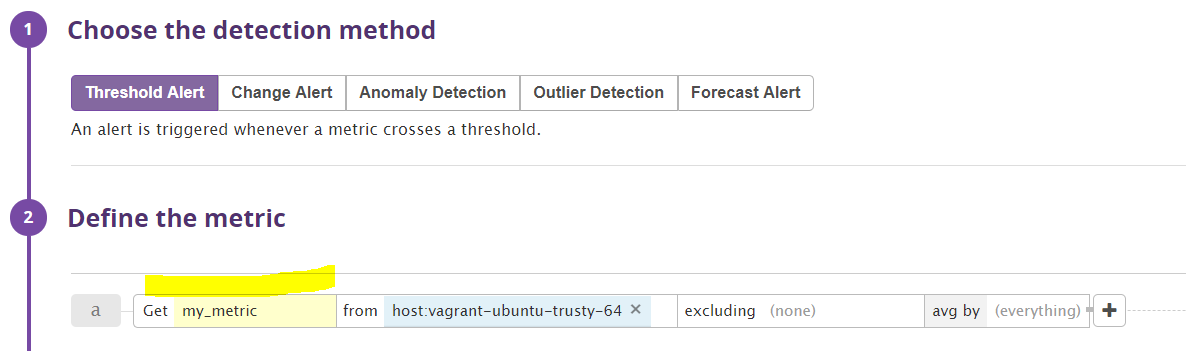
By analysing a metric’s historical behaviour, anomaly detection distinguishes between normal and abnormal metric trends. The anomaly graph above for MySQL on my Ubuntu Linux server is showing the sum of the CPU performance metric over a 1-hour period.

# Exercise 3: Monitoring Data

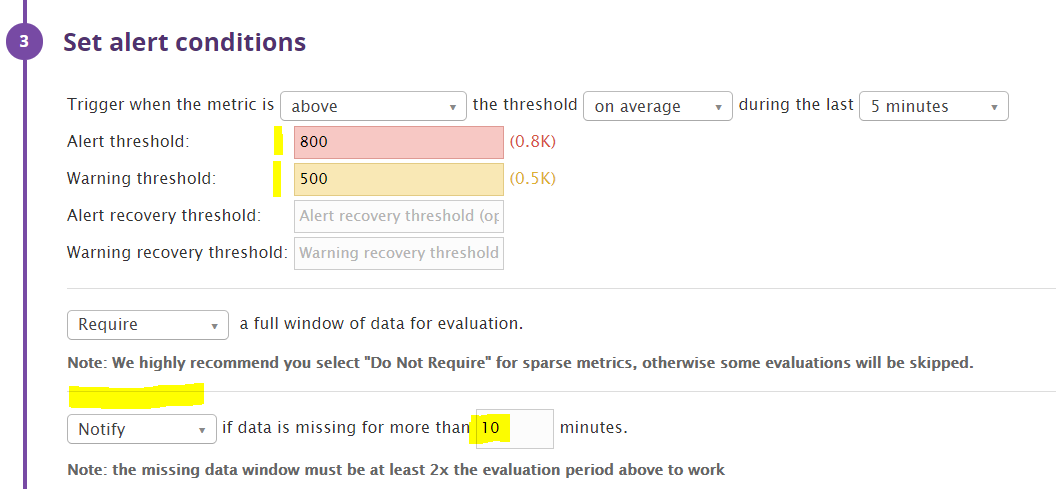
The third task was to set up a Threshold Alert for my\_metric. From selecting the Monitors section on the left of the UI, I then went to add a new monitor from the top:



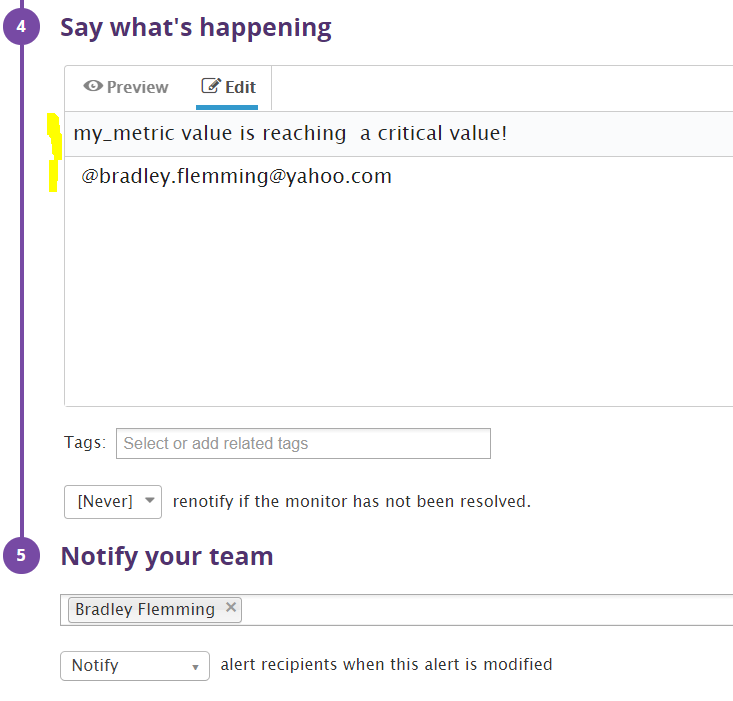
Then it was a case of setting up a new Threshold alert as below shows:



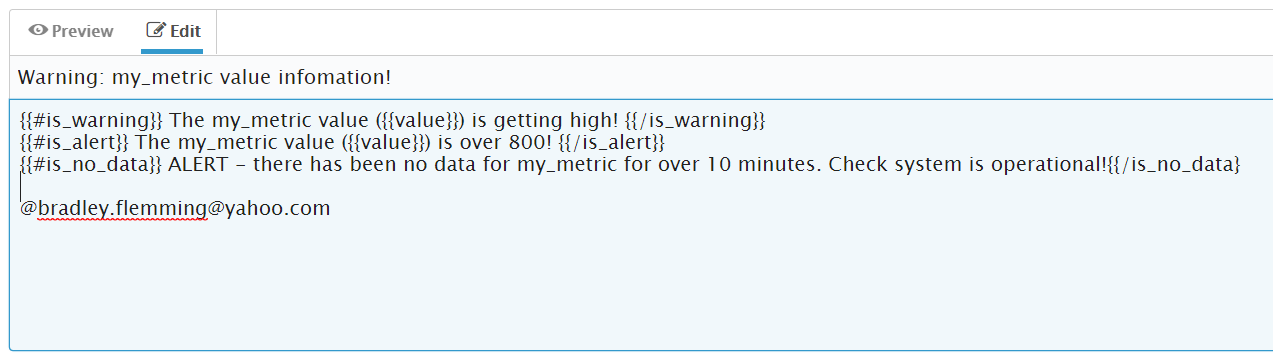
For the Alert I have configured a Warning set at the value of 500 and an Alert at the value of 800. There is also a Notification set so when there is no data detected for more than 10 minutes as below:



Who will be alerted – me! When my\_metric reaches the thresholds, I will be alerted with a relevant message via e-mail:



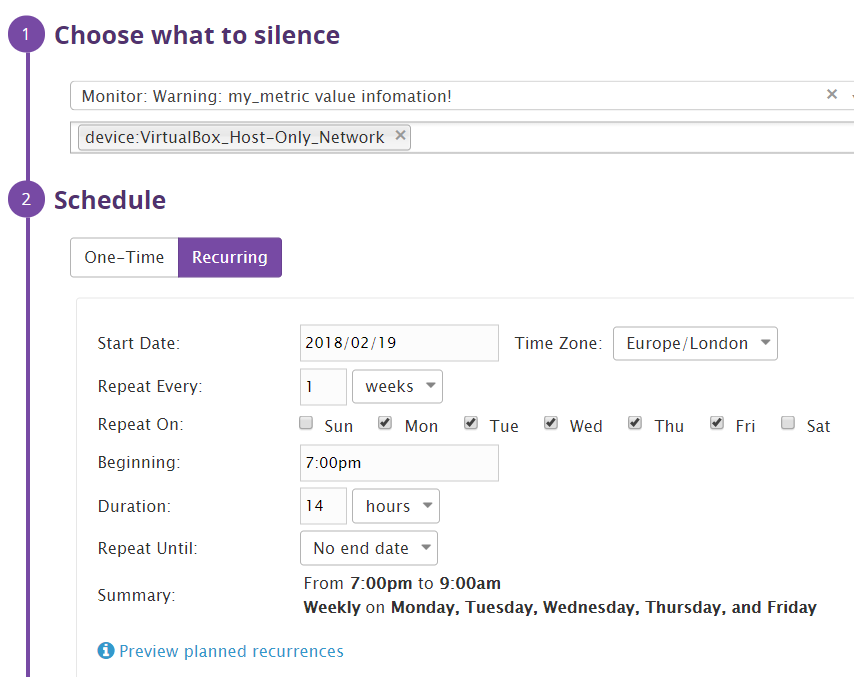
Below is the script I have written to send a different message depending on the Threshold value:



Below is an example of the Notification e-mail I received:



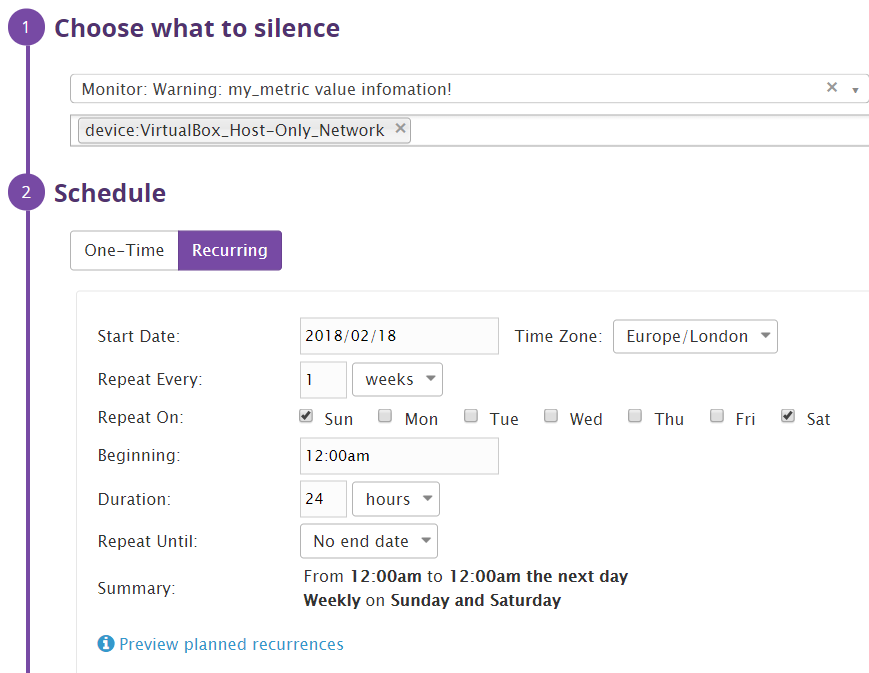
We can also schedule downtime between 7pm and 9am Mon – Fri every week. This prevents unnecessary monitoring if you know the applications will be rarely used during this timeframe. We can set DataDog to notify us of this in the event we need to enable monitoring for a particular reason:



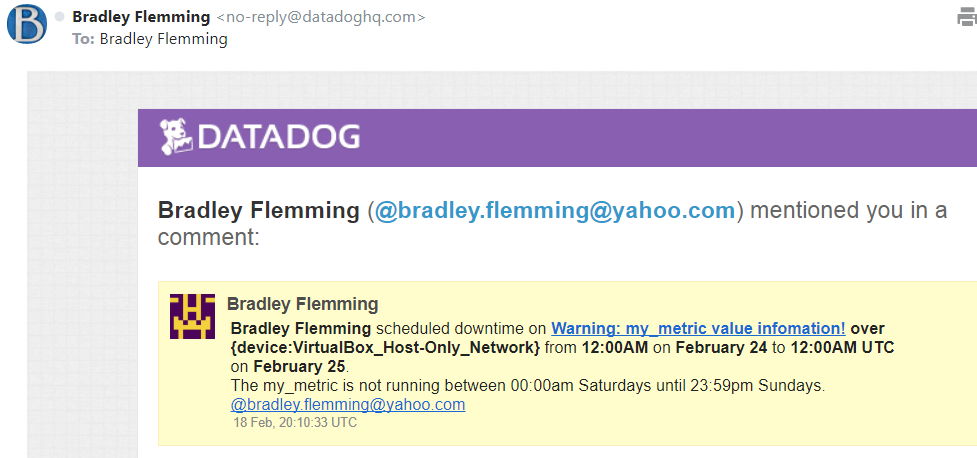
Below is an example of the e-mail we would receive:



We can also schedule downtime between 12am Saturday morning until midnight Sunday evening, every weekend. Configured as:

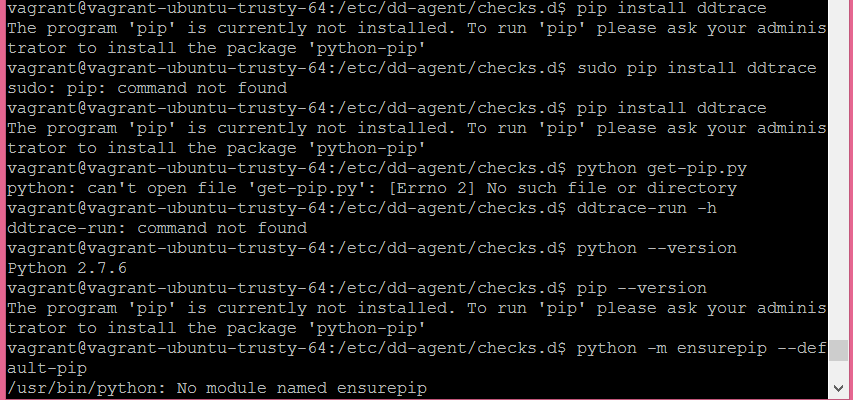


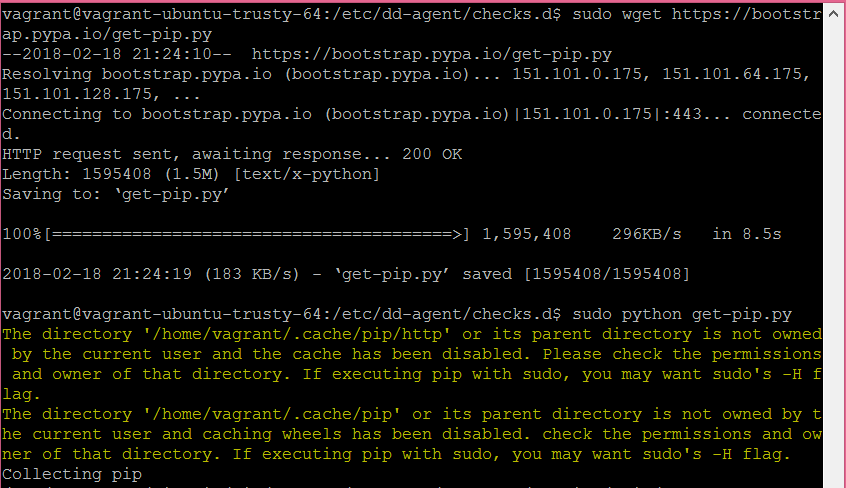
Again – here’s what the notification e-mail looks like so we are reminded there is no monitoring taking place during this timeframe:

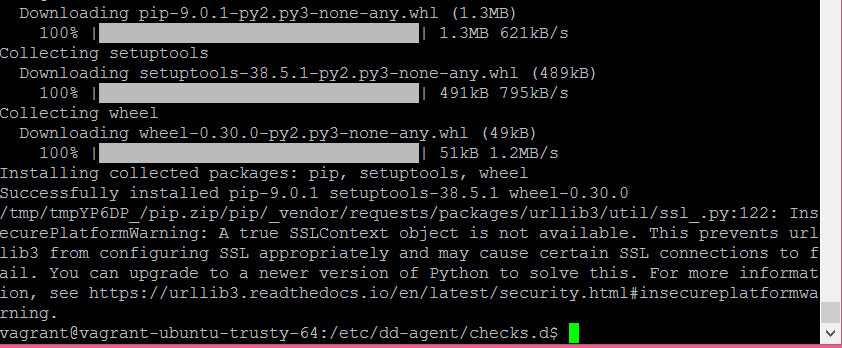


# Exercise 4: Collecting APM Data

In trying to complete this last part of the Lab I discovered Pip for python was not installed nor available on the VM – I had to download this separately, upgrade and install it:

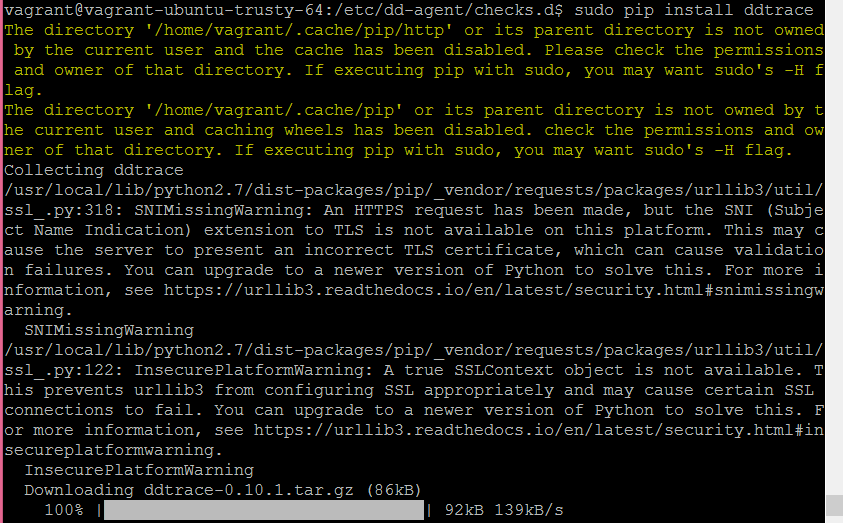




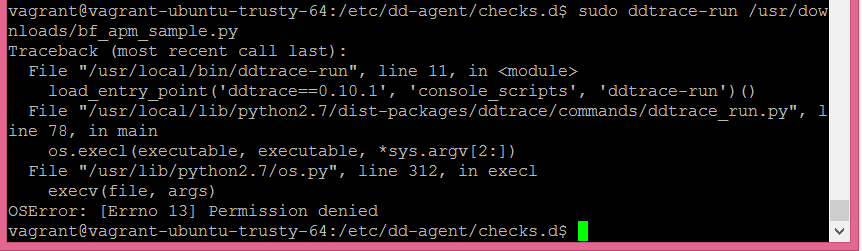


Once we have this in place we could then install ‘ddtrace’. This is required so we can still monitor applications or custom code that may have been developed specifically by an organisation to meet a specific business need that the Datadog list of integrations does not support. Typically, these monitor app’s written in Java, Python, Ruby, Go, or many other common development languages.

I could download ddtrace:



…BUT I COULDN’T GET APM TRACING TO RUN !!



Failure of this last task was very frustrating! To appreciate my level of frustration you might want to double-click the image below…



What is the difference between a Service and a Resource?

* Service: the name of a set of processes that work together to provide a feature set.
* Resource: A particular query to a service.

# Final Question – APM Use Case

London has a congestion charge – how well does it implement this?

|  |  |
| --- | --- |
|  | There’s the question – how well has the City of London implemented its congestion charging policy, how does it know and what else could they do with that data?  In order for Transport For London to be able to answer this question they need data, they need a lot of it, they need it from a lot of different sources and they need to be able to correlate the data into meaningful information.  From a consumer/user perspective we can say this is a monetising exercise for the city – how do we make money from all of these vehicles coming into the city with polluting vehicles in a way that makes us look good? Answer – call it a congestion charge. The result is fewer vehicles driving around London thus lowering the amount of emissions from combustion engines.  Great Idea! |
| Q. How will we know if it’s working?  A. We get revenue!  Q. How much revenue?  A. Lots! There are loads of cars, buses, taxis, motorcycles, etc. We’ll make them pay for using the roads.  Q. How do you know they’ve paid?  A. Hmmm… Let me get back to you on that.  So, you know where this is going – we monitor the data from in-road traffic sensors, traffic camera recognition technology, number plate recognition technology, DVLA database integration (is the car registered, is it insured) etc, etc. There’s a lot of existing infrastructure collecting a lot of data about vehicles, the number of vehicles, moving vehicles, speed of moving vehicles, stationary vehicles, clean vehicles, dirty vehicles, etc.  So – can they now consider traffic re-direction away from congested areas, can they use the same sensors to predict slow buses that might be running behind schedule, can they improve air quality by allowing free-flowing traffic or no traffic in certain areas?  How would we do that? Well – with all that data we could put a variable road charge into force, cheaper charge for routes less travelled, higher charges for polluting roads, school roads, congested roads. This is just usage of vehicles on roads, augment this with train information, underground information, we start to reveal the benefits of a truly connected transport infrastructure.  Not only does this provide an almost immediate response to unusual situations, but then consider performing predictive maintenance on the roads rather than responding to excessive insurance claims, and what’s under the roads. In London, that’s a lot. | |