Table of Contents

[Collecting Metrics: 1](#_Toc507590243)

[Data visualisation 5](#_Toc507590244)

[Monitoring Data 8](#_Toc507590245)

[Collecting APM Data 9](#_Toc507590246)

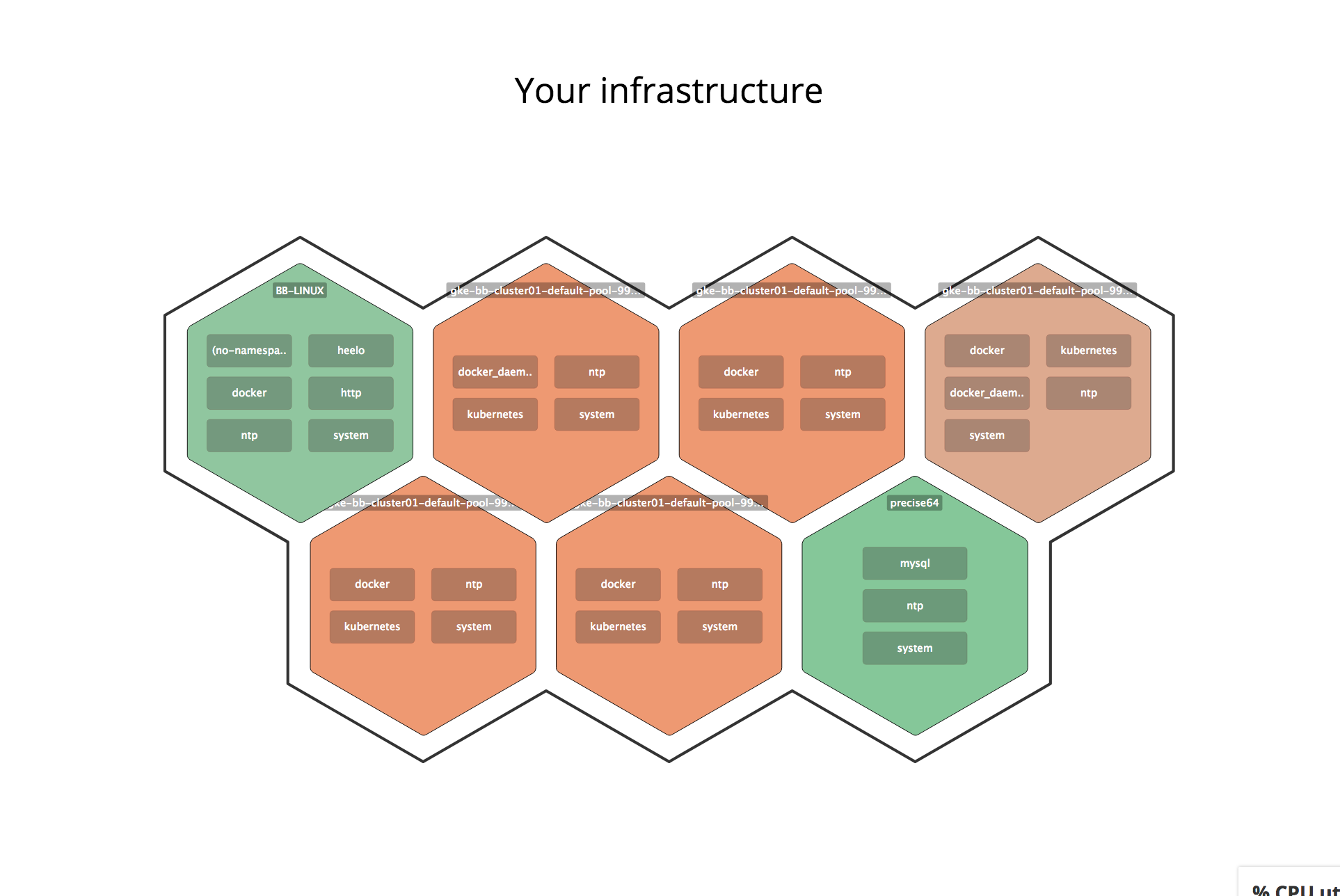
[Final Question: 9](#_Toc507590247)

# Collecting Metrics:

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.

**Host Map**

**I created multiple with Vagrant, GCP docker, kubernetes to play with the metrics..**



**Agent Config**

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

# hostname: mymachine.mydomain

# Enable the trace agent.

# apm\_enabled: false

# Enable gohai metadata collection

# enable\_gohai: true

# Enable the process agent

# process\_agent\_enabled: false

# Set the host's tags (optional)

tags: region:UK, env:home\_server, os:ubuntu, running:docker

# Collect docker orchestrator name and version as agent tags

# collect\_orchestrator\_tags: yes

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

**MySQL Agent config**

# NOTE: Even if the server name is "localhost", the agent will connect to MySQL using TCP/IP, unless you also

# provide a value for the sock key (below).

- server: localhost

user: datadog

pass: EJK7wMGJjiqWiwkF7Yuehrs\_

# port: 3306 # Optional

# sock: /path/to/sock # Connect via Unix Socket

# defaults\_file: my.cnf # Alternate configuration mechanism

# connect\_timeout: None # Optional integer seconds

tags:

- mysqlbb

options: # Optional

replication: 0

# replication\_channel: channel\_1 # If using multiple sources, the channel name to monitor

# replication\_non\_blocking\_status: false # grab slave count in non-blocking manner (req. performance\_schema)

galera\_cluster: 1

# extra\_status\_metrics: true

# extra\_innodb\_metrics: true

# extra\_performance\_metrics: true

# schema\_size\_metrics: false

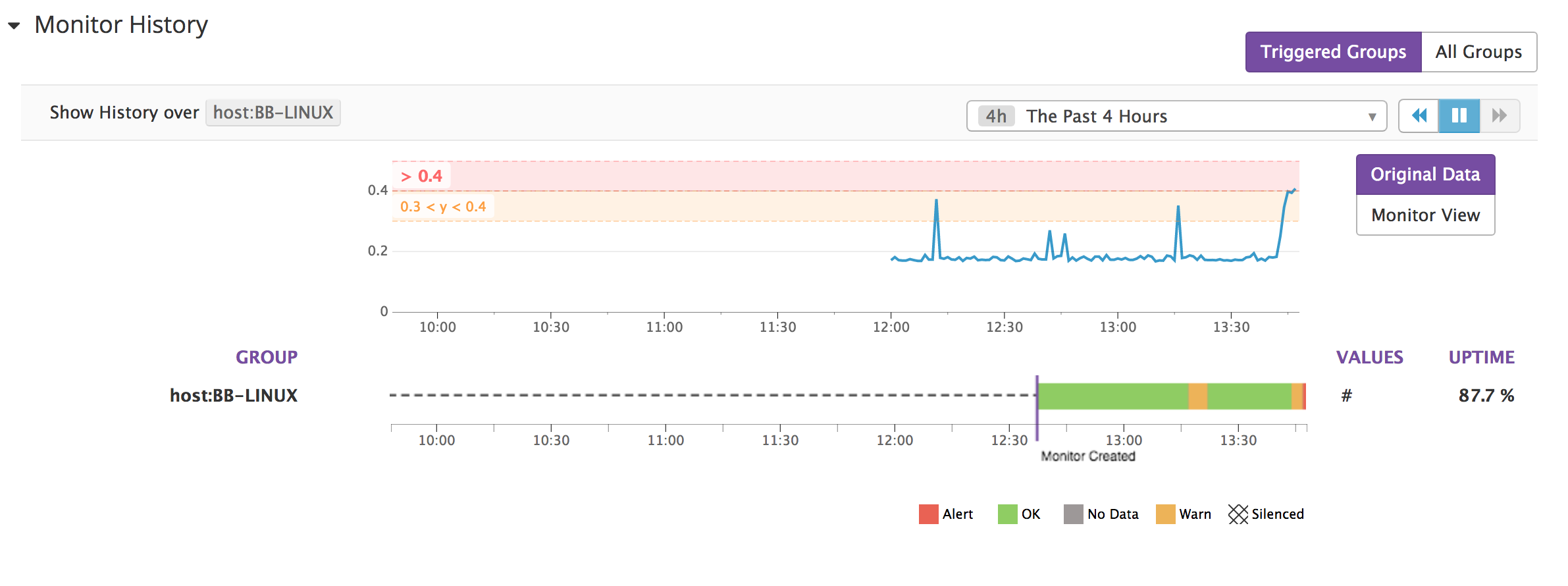
# disable\_innodb\_metrics: false

#

Create a custom Agent check that submits a metric named my\_metric with a random value between 0 and 1000.

Created a http agent check, from documentation which submits the http response time as a random value between 0 and 1000

**http response**

****

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

Updated the my\_metric.yaml file to set min\_collection\_interval value to 45

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

Yes, see screenshot below:

**YAML File**

init\_config:

default\_timeout: 5

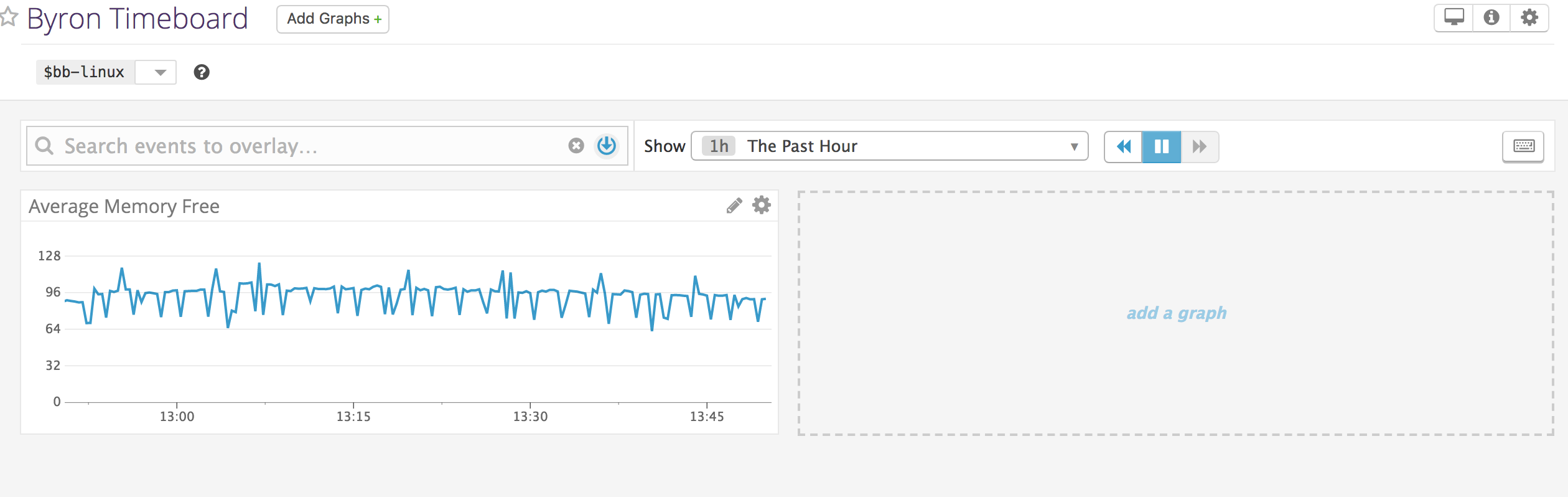
min\_collection\_interval: 45

# Data visualisation

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

**Timeboard image**



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

Please see the script below

*from datadog import initialize, api*

*options = {*

*'api\_key': '388fd7f864f4f41fe1bca10e422ff1a6',*

*'app\_key': '1d675c154b2cc8d41cf4300a2a511ea9e835fd9e'*

*}*

*initialize(\*\*options)*

*title = "Byron Timeboard"*

*description = "my first attempt using the API."*

*graphs = [{*

*"definition": {*

*"events": [],*

*"requests": [*

*{"q": "avg:system.mem.free{\*}"}*

*],*

*"viz": "timeseries"*

*},*

*"title": "Average Memory Free"*

*}]*

*template\_variables = [{*

*"name": "bb-linux",*

*"prefix": ,*

*"default":*

*}]*

*read\_only = True*

*api.Timeboard.create(title=title,*

*description=description,*

*graphs=graphs,*

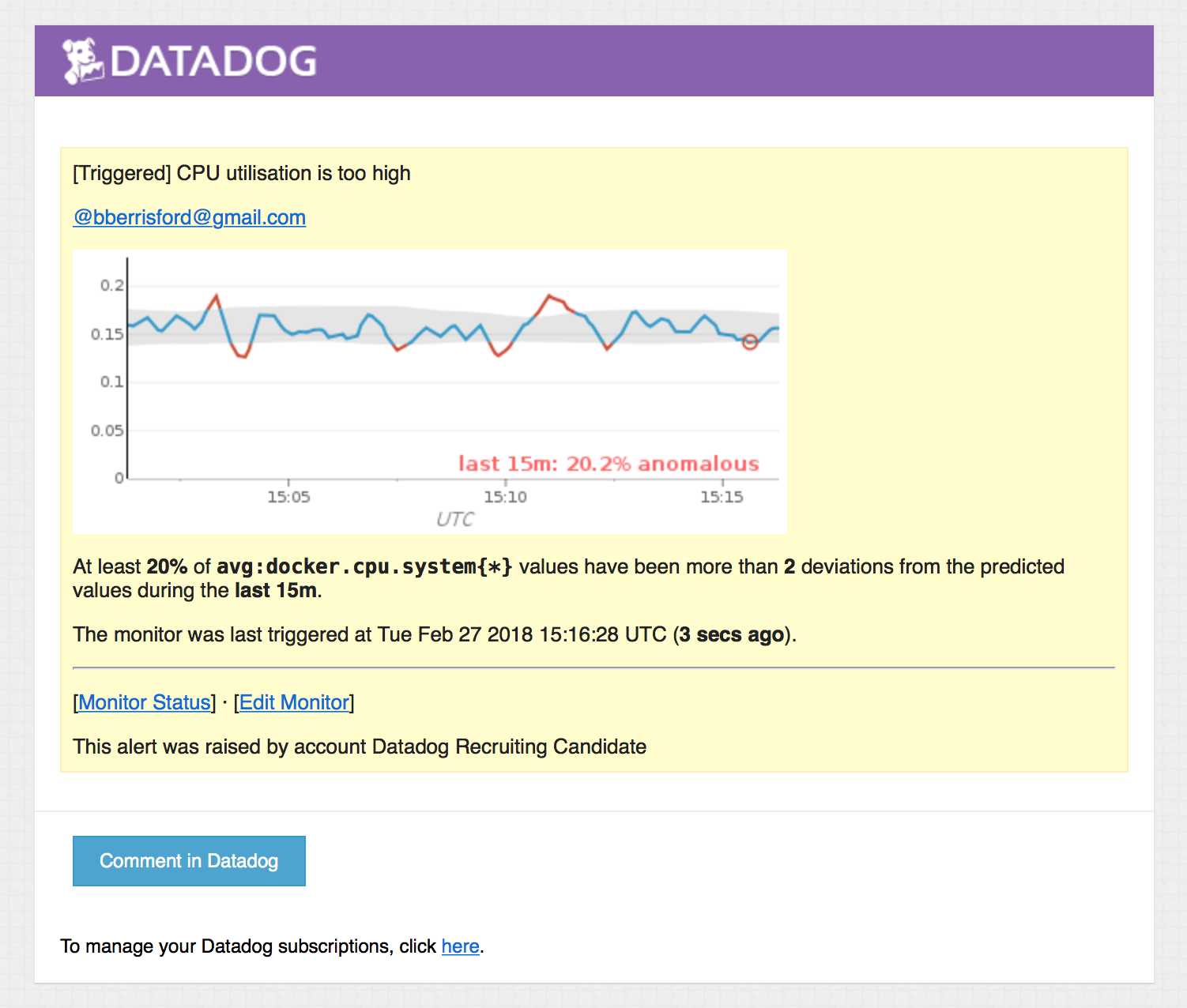
*template\_variables=template\_variables,*

*read\_only=read\_only)*

Once this is created, access the Dashboard from your Dashboard List in the UI:

* Set the Timeboard's timeframe to the past 5 minutes
  + Its not possible to set timeframe to last 5 minuets, the filter only go as far back as 1hour
* Take a snapshot of this graph and use the @ notation to send it to yourself.

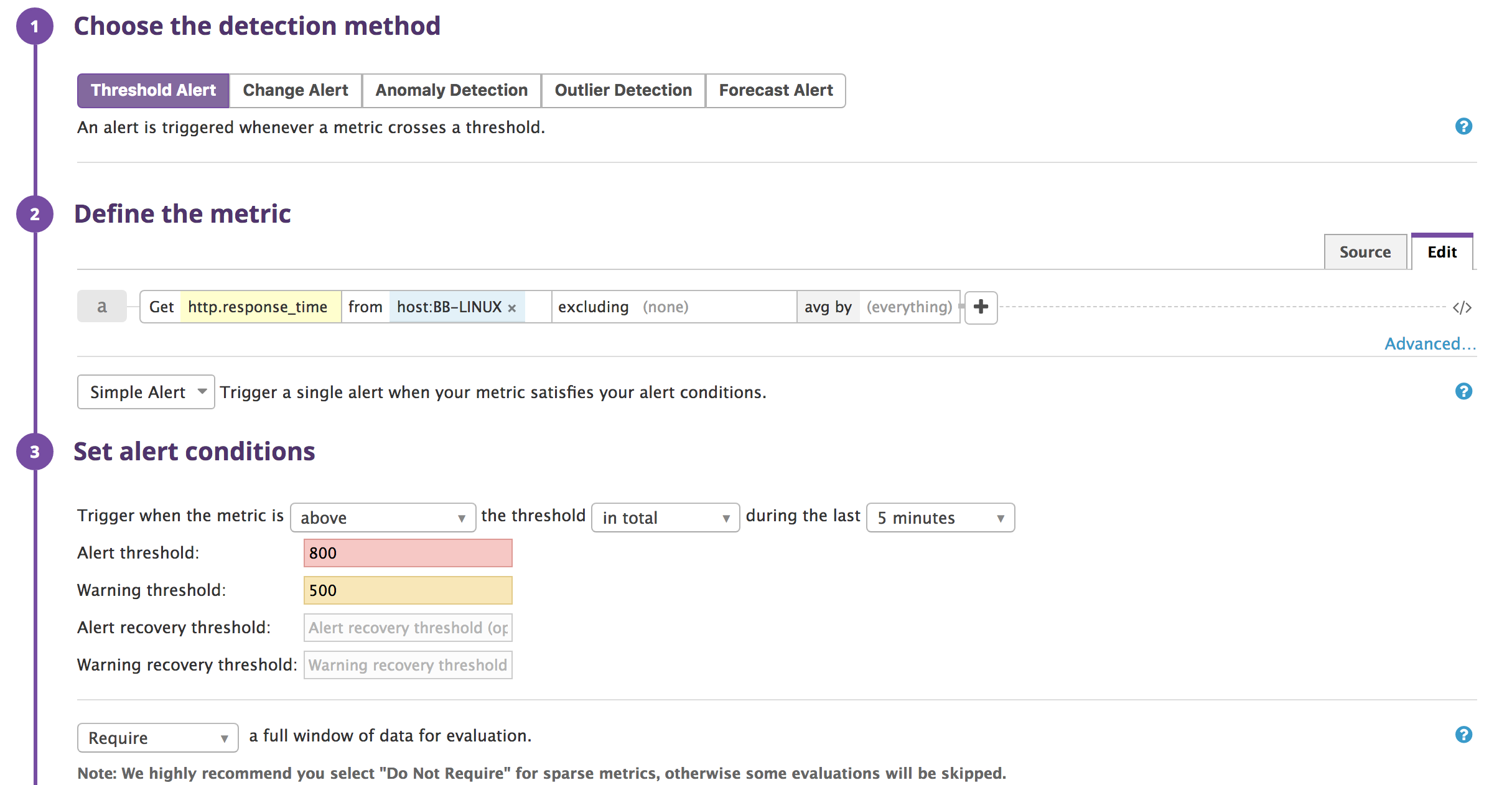
**Email trigger**

****

# Monitoring Data

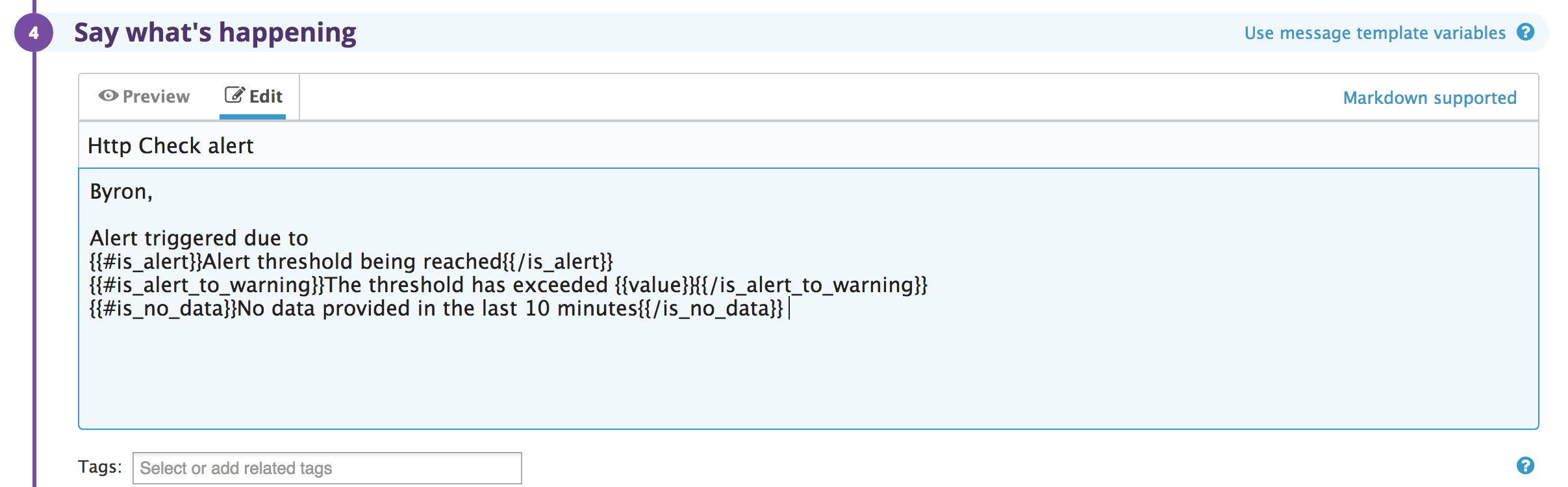
Create a new Metric Monitor that watches the average of your custom metric (my\_metric) and will alert if it’s above the following values over the past 5 minutes:

* 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

****

# Collecting APM Data

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

The provided dashboard below has the following aim, it lists the total customer visits and website reponse times over the last hour.

The dashboard also maps infrastructure/system utilisation against the application, such that its easy to debug slow response times against database, cpu and memory utilisation.

Created APM python script but for some reason it never materialised in the APM monitor

I’ve attached the script under myflaskapp.py

I used the provided flask application and the ddtrace-run python agent.

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

A resource is a a particular query or criteria attached to the service, whilst a service is a named set of resources which form a feature.

# Final Question:

Is there anything creative you would use Datadog for?

Datadog with all of its metrics and custom agents can be the trigger to kick of additional microservices, for instance if the a webapp response time is diminishing and additionally PCU across the app is high etc, then new front end services can be deployed to load balance the requests, this could be verified with the qty of API requests