



# DATADOG

Hiring Engineers

*Efrain Gamboa*

## Table of Contents

<b>Prerequisites - Setup the environment.....</b>	<b>2</b>
Create an instance in AWS .....	2
Installing the agent .....	2
<b>Collecting Metrics.....</b>	<b>4</b>
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.....	4
Install a database on your machine (MongoDB, MySQL, or PostgreSQL) and then install the respective Datadog integration for that database.....	5
Create a custom Agent check that submits a metric named my_metric with a random value between 0 and 1000 .....	6
Change your check's collection interval so that it only submits the metric once every 45 seconds.....	7
Bonus Question .....	8
<b>Visualizing Data.....</b>	<b>9</b>
Utilize the Datadog API to create a Timeboard.....	9
Once this is created, access the Dashboard from your Dashboard List in the UI .....	14
Bonus Question .....	14
<b>Monitoring Data.....</b>	<b>15</b>
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.....	15
Bonus Question .....	19
<b>Collecting APM Data.....</b>	<b>20</b>
Initial Setup.....	21
APM Setup .....	22
Bonus Question .....	25
<b>Final Question .....</b>	<b>25</b>
<b>Extra Information .....</b>	<b>26</b>

## Prerequisites - Setup the environment

Create an instance in AWS

I deployed an t2.micro EC2 instance with os: Ubuntu 20.04 (focal) in the London region, created a new key pair for this instance to be able to SSH into it. While creating the instance I created a security group and allow SSH traffic on port 22. And enabled the auto-assign for public IP.

Instance summary for i-0a21888e802e5a440 (Efrain) <a href="#">Info</a>		
Updated less than a minute ago		
Instance ID i-0a21888e802e5a440 (Efrain)	Public IPv4 address 35.176.71.115   <a href="#">open address</a>	Private IPv4 addresses 172.31.41.67
Instance state <span>Running</span>	Public IPv4 DNS ec2-35-176-71-115.eu-west-2.compute.amazonaws.com   <a href="#">open address</a>	Private IPv4 DNS ip-172-31-41-67.eu-west-2.compute.internal
Instance type t2.micro	Elastic IP addresses -	VPC ID vpc-48d14320 (Side-Car Connected VPC)
AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations.   <a href="#">Learn more</a>	IAM Role -	Subnet ID subnet-68ff0824

## Installing the agent

To installed the agent I accessed the EC2 instance via SSH. And followed the datadog agent installation guide from the web interface. Which is executing the following command in the console

```
DD_AGENT_MAJOR_VERSION=7 DD_API_KEY=<API_KEY> DD_SITE="datadoghq.eu" bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install_script.sh)"
```

```
Downloads — ubuntu@ip-172-31-41-67: ~ — ssh -i EG-London.p...
* Adding your API key to the Agent configuration: /etc/datadog-agent/datadog.yaml
* Setting SITE in the Agent configuration: /etc/datadog-agent/datadog.yaml
/usr/bin/systemctl
* Starting the Agent...

Your Agent is running and functioning properly. It will continue to run in the
background and submit metrics to Datadog.

If you ever want to stop the Agent, run:

    sudo systemctl stop datadog-agent

And to run it again run:

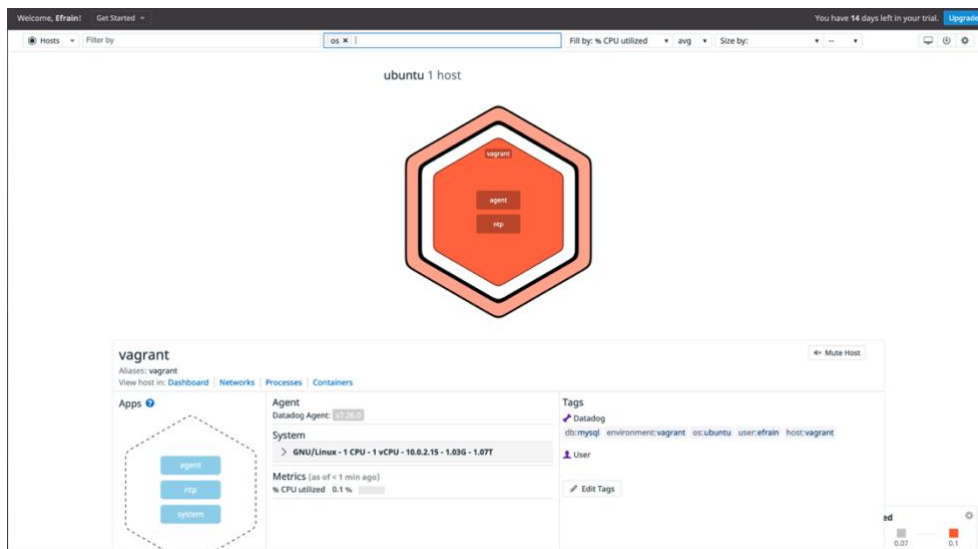
    sudo systemctl start datadog-agent

ubuntu@ip-172-31-41-67:~$
```

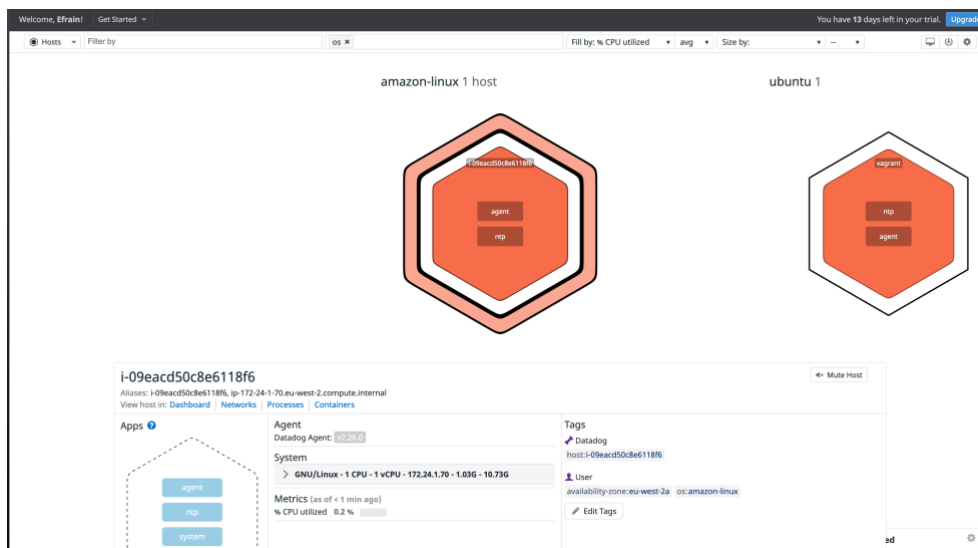
I also deployed a couple other instances to try different things.

- VM via Vagrant – Ubuntu 20.10
- VM via Fusion – Ubuntu 20.10
- AWS EC2 – Amazon Linux

I added some tags to the vagrant Ubuntu machine directly in the configuration file, and then added some tags to the EC2 linux machine via the UI to see the difference.



Vagrant machine with tags added in the Agent's config file



EC2 instance with tags added directly in the UI

## 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.

From the datadog documentation: There are a few different ways to add tags to your instances.

### Assigning tags

#### Tagging methods

Tags may be assigned using any (or all) of the following methods. Refer to the dedicated [Assigning Tags documentation](#) to learn more:

METHOD	ASSIGN TAGS
<a href="#">Configuration Files</a>	Manually in your main Agent or integration configuration files.
<a href="#">UI</a>	In the Datadog App
<a href="#">API</a>	When using Datadog's API
<a href="#">DogStatsD</a>	When submitting metrics via DogStatsD

For this exercise I have followed the “Configuration Files” method, which consists on adding the tags directly in the datadog agent’s configuration file.

The configuration file for Ubuntu are under the following path:  
/etc/datadog-agent/datadog.yaml

```
datadog — vagrant@vagrant: /etc/datadog-agent — ssh • vagran...
## FQDN instead of the short hostname. Recommended value: true
## More information at https://dtdg.co/flag-hostname-fqdn
#
# hostname_fqdn: false

## @param tags - list of key:value elements - optional
## List of host tags. Attached in-app to every metric, event, log, trace, and service check emitted by this Agent.
##
## Learn more about tagging: https://docs.datadoghq.com/tagging/
#
tags:
  - environment:vagrant
  - os:ubuntu
  - version:18.04
  - user:egamboa
  - db:mongo
  - db:mysql

## @param env - string - optional
## The environment name where the agent is running. Attached in-app to every metric, event, log, trace, and service check emitted by this Agent.
#
-- INSERT --                                     72,14      3%
```

After adding the tags it is necessary to restart the agent.  
sudo service datadog-agent restart

Then I ran the agent status check to verify that the changes had taken place  
sudo datadog-agent status

```
Downloads — ubuntu@ip-172-31-41-67: /etc/datadog-agent — s...

Hostnames
=====
ec2-hostname: ip-172-31-41-67.eu-west-2.compute.internal
hostname: i-0a21888e802e5a440
instance-id: i-0a21888e802e5a440
socket-fqdn: ip-172-31-41-67.eu-west-2.compute.internal.
socket-hostname: ip-172-31-41-67
host tags:
  environment: aws
  user: egamboa
  os: ubuntu
  db: mongodb
  db: mysql
hostname provider: aws
unused hostname providers:
  configuration/environment: hostname is empty
  gce: unable to retrieve hostname from GCE: status code 404 trying to GET h
ttp://169.254.169.254/computeMetadata/v1/instance/hostname

Metadata
=====
cloud_provider: AWS
hostname_source: aws
```

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

I decided to install mysql for ubuntu:

```
sudo apt-get install mysql-server
```

And then installed the integration following the mysql installation guide. In this case my mysql deployment is version 8.0.23, so the steps required were:

```
mysql> CREATE USER 'datadog'@'localhost' IDENTIFIED WITH
mysql_native_password by '<UNIQUEPASSWORD>';
```

```
mysql> ALTER USER 'datadog'@'localhost' WITH MAX_USER_CONNECTIONS 5;
```

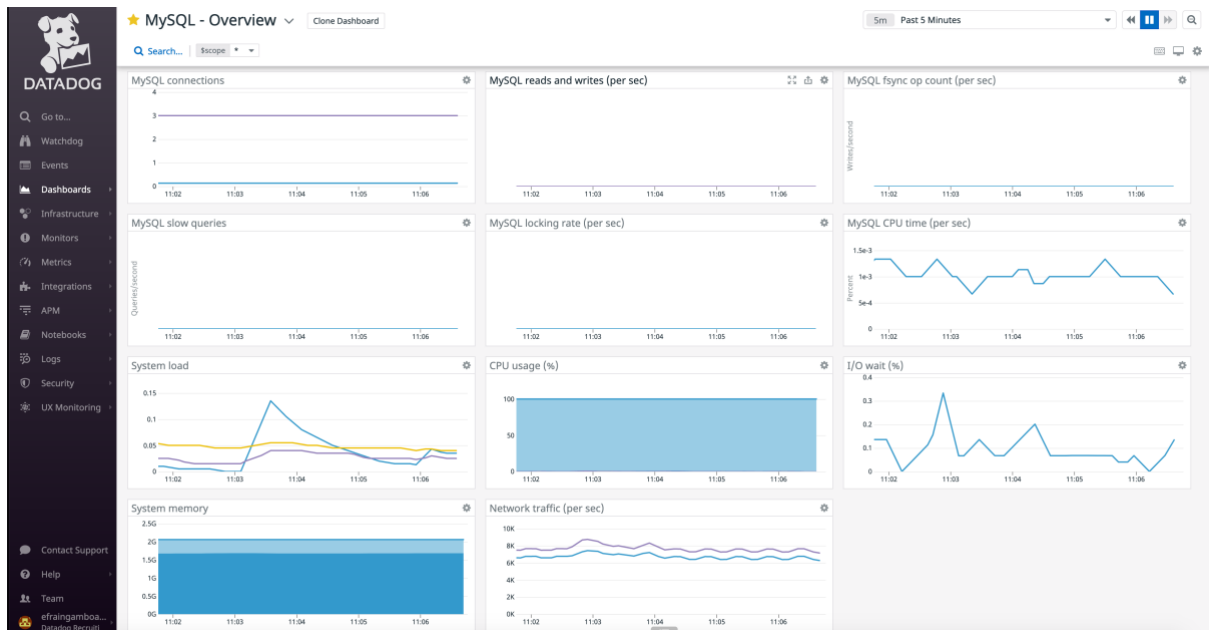
```
Downloads — ubuntu@ip-172-31-41-67: /etc/datadog-agent/conf.d/mysql.d — ssh...

mysql (2.3.0)
-----
Instance ID: mysql:20a3eabb46b86ae7 [WARNING]
Configuration Source: file:/etc/datadog-agent/conf.d/mysql.d/conf.yaml
Total Runs: 1
Metric Samples: Last Run: 117, Total: 117
Events: Last Run: 0, Total: 0
Service Checks: Last Run: 1, Total: 1
Average Execution Time : 36ms
Last Execution Date : 2021-03-28 09:03:30.000000 UTC
Last Successful Execution Date : 2021-03-28 09:03:30.000000 UTC
metadata:
  flavor: MySQL
  version.build: unspecified
  version.major: 8
  version.minor: 0
  version.patch: 23
  version.raw: 8.0.23+unspecified
  version.scheme: semver

Warning: Privileges error accessing the BINARY LOGS (must grant REPLICATION CLIENT): (1227,
'Access denied; you need (at least one of) the SUPER, REPLICATION CLIENT privilege(s) for this ope
ration')
```

Then restarted the agent and ran the agent status check.

In the console then navigated to “Dashboards” and looked for the “MySQL – Overview” Dashboard and verified that data was being gathered.



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

In order to create a custom metric there are also a few available methods to send metrics to Datadog:

- Custom Agent Check
- DogStatsD
- PowerShell
- AWS Lambda
- Datadog's HTTP API

For this exercise it was requested to create a custom agent check, and in order to create this custom check in the agent I followed the documentation at:

[https://docs.datadoghq.com/developers/metrics/agent\\_metrics\\_submission/?tab=count](https://docs.datadoghq.com/developers/metrics/agent_metrics_submission/?tab=count)

```
Downloads — ubuntu@ip-172-31-41-67: /etc/datadog-agent/checks.d — ssh -i EG-...
import random

from datadog_checks.base import AgentCheck

__version__ = "1.0.0"

class MyClass(AgentCheck):
    def check(self, instance):

        # Calling the functions below twice simulates
        # several metrics submissions during one Agent run.
        self.gauge(
            "my_metric.gauge",
            random.randint(0, 1000),
            tags=["env:dev", "metric_submission_type:gauge"],
        )
        self.gauge(
            "my_metric.gauge",
            random.randint(0, 1000),
            tags=["env:dev", "metric_submission_type:gauge"],
        )

"my_metric.py" 22L, 579C                                     17,19      All
```

Restarted the agent and ran the agent status check, and looked for the newly created custom agent check.

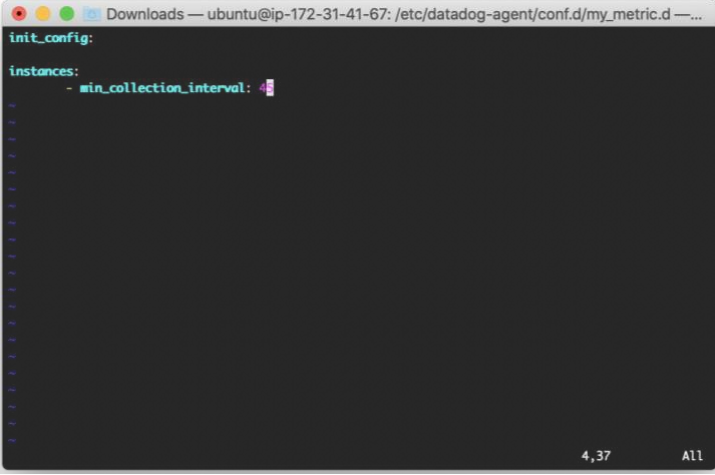
```
Downloads — ubuntu@ip-172-31-41-67: /etc/datadog-agent/checks.d — ssh -i EG-...
memory
-----
Instance ID: memory [OK]
Configuration Source: file:/etc/datadog-agent/conf.d/memory.d/conf.yaml.default
Total Runs: 1
Metric Samples: Last Run: 18, Total: 18
Events: Last Run: 0, Total: 0
Service Checks: Last Run: 0, Total: 0
Average Execution Time : 0s
Last Execution Date : 2021-03-28 09:19:44.000000 UTC
Last Successful Execution Date : 2021-03-28 09:19:44.000000 UTC

my_metric (1.0.0)
-----
Instance ID: my_metric:d884b5186b651429 [OK]
Configuration Source: file:/etc/datadog-agent/conf.d/my_metric.d/my_metric.yaml
Total Runs: 1
Metric Samples: Last Run: 2, Total: 2
Events: Last Run: 0, Total: 0
Service Checks: Last Run: 0, Total: 0
Average Execution Time : 0s
Last Execution Date : 2021-03-28 09:19:40.000000 UTC
Last Successful Execution Date : 2021-03-28 09:19:40.000000 UTC
```

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

In `my_metric.d/` folder, modify the configuration file named `my_metric.yaml` with the collection interval set to 45 seconds





```
Downloads — ubuntu@ip-172-31-41-67: /etc/datadog-agent/conf.d/my_metric.d — ...
init_config:

instances:
  - min_collection_interval: 4
```

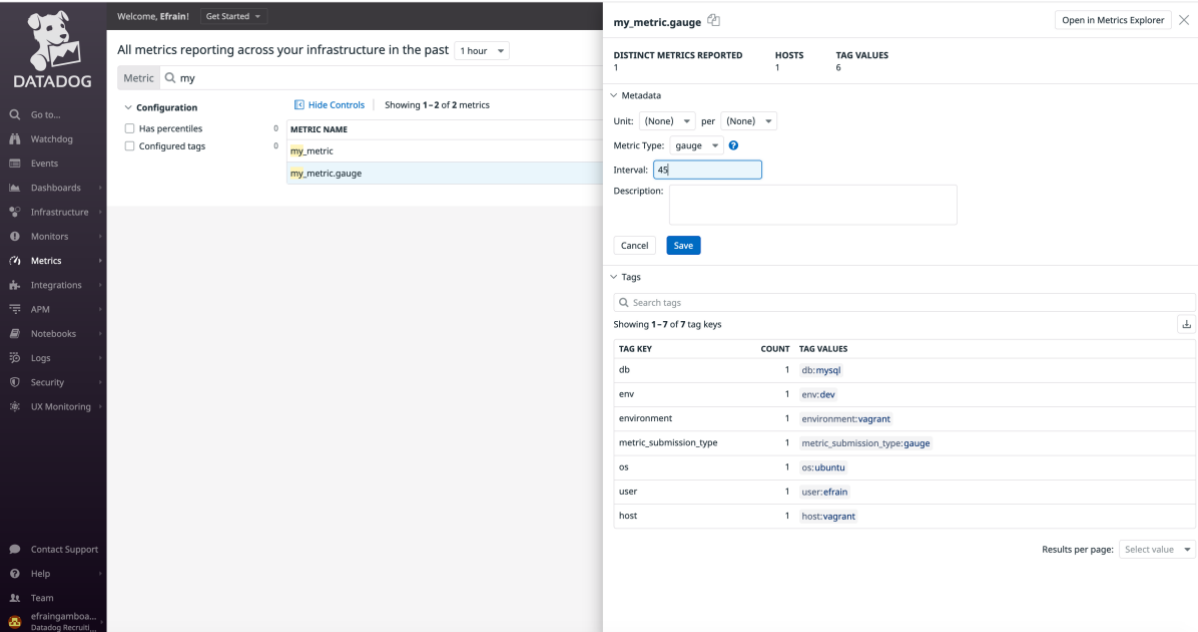
**Bonus Question:** Can you change the collection interval without modifying the Python check file you created?

Yes. It is possible to change the collection interval in two possible ways. The first one being modifying the python check file, as done in the previous step.

Or alternatively: It can also be changed directly the Datadog UI.

Procedure:

- Navigate to metrics -> Summary
- In the search bar input the name of the metric
- Then in the metadata section click on “Edit”, and modify the interval



my\_metric.gauge Open in Metrics Explorer

DISTINCT METRICS REPORTED	HOSTS	TAG VALUES
1	1	6

Unit: (None) per (None)

Metric Type: gauge

Interval: 45

Description:

Tags

TAG KEY	COUNT	TAG VALUES
db	1	db:mysql
env	1	env:dev
environment	1	environment:vagrant
metric_submission_type	1	metric_submission_type:gauge
os	1	os:ubuntu
user	1	user:efrain
host	1	host:vagrant

Results per page: Select value

## Visualizing Data

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

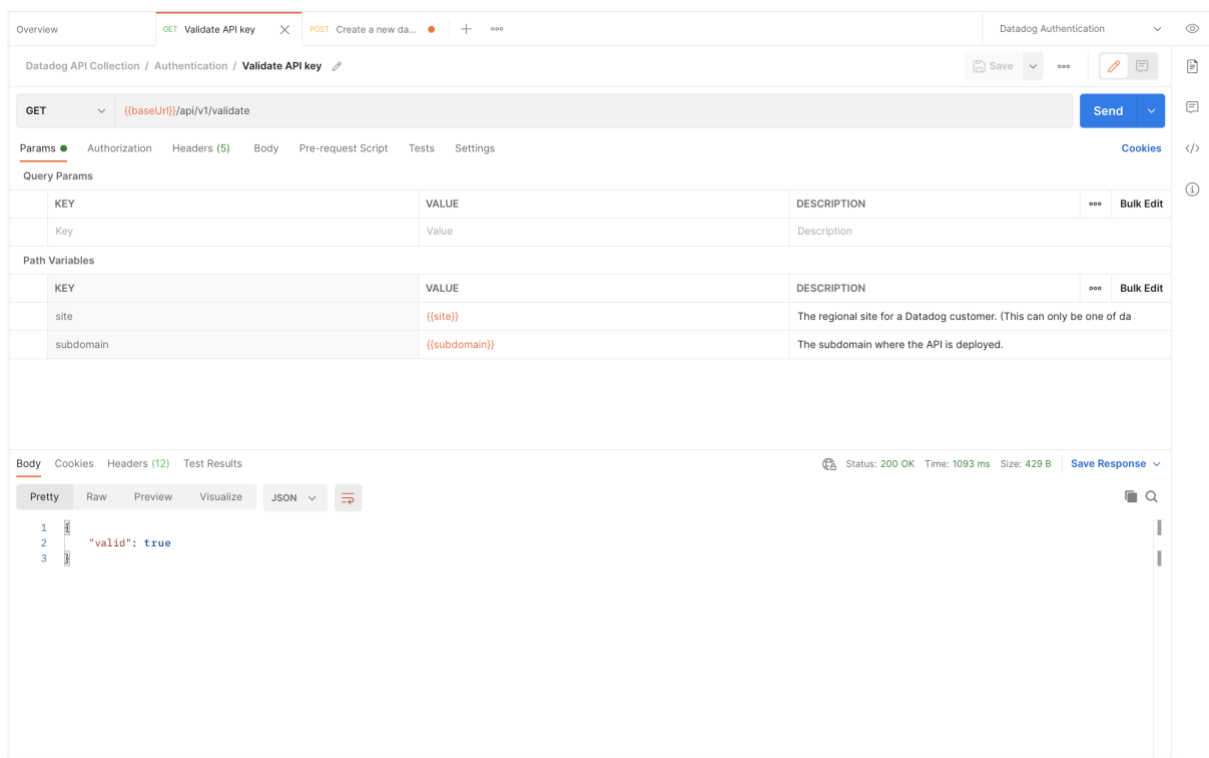
### Initial setup

Documentation about the API

<https://docs.datadoghq.com/api/latest/>

I set up postman and imported the collection. And set the the api\_key and application\_key in the “Datadog Authentication” environment.

Then validated the key and that I was able to reach the service using the Authentication call.



Now that the it is all set up and I am able to interact with the API the next step it's to create the the Timeboard. For this I followed the documentation at:

<https://docs.datadoghq.com/api/latest/dashboards/>

To create a new dashboard it is necessary to use the following API call:

POST <https://api.datadoghq.eu/api/v1/dashboard>

And as guide there is the following Example JSON:

```
{
  "description": "string",
  "is_read_only": false,
```

```
"layout_type": "string",
"notify_list": [],
"template_variable_presets": [
  {
    "name": "string",
    "template_variables": [
      {
        "name": "string",
        "value": "string"
      }
    ]
  }
],
"template_variables": [
  {
    "default": "my-host",
    "name": "host1",
    "prefix": "host"
  }
],
"title": "",
"widgets": [
  {
    "definition": "[object Object]",
    "id": "integer",
    "layout": {
      "height": 0,
      "width": 0,
      "x": 0,
      "y": 0
    }
  }
]
}
```

This is an example of how the body of the request should look like. Not being familiar with the tool I was not sure of the possibilities or what to put in the JSON in order to reflect what is being request. So I went to the datadog UI and used to look at creating the dashboard to export the JSON. Which I then added to the body of the request, and successfully created via API.

- [Your custom metric scoped over your host.](#)

```
{
  "viz": "timeseries",
  "requests": [
    {
      "q": "max:my_metric{*}",
      "type": "line",
      "style": {
        "palette": "purple",
        "type": "solid",
        "width": "thin"
      },
      "on_right_yaxis": false
    }
  ],
  "yaxis": {
    "scale": "linear",
    "min": "auto",

```

```
    "max": "auto",
    "includeZero": true,
    "label": ""
  },
  "markers": []
}
```

- [Any metric from the Integration on your Database with the anomaly function applied](#)

**Metric: mysql.performance.cpu\_time**

```
{
  "viz": "timeseries",
  "requests": [
    {
      "q": "anomalies(avg:mysql.performance.cpu_time{*}, 'basic',
2) ",
      "type": "line",
      "style": {
        "palette": "purple",
        "type": "solid",
        "width": "normal"
      },
      "on_right_yaxis": false
    }
  ],
  "yaxis": {
    "scale": "linear",
    "min": "auto",
    "max": "auto",
    "includeZero": true,
    "label": ""
  },
  "markers": []
}
```

- [Your custom metric with the rollup function applied to sum up all the points for the past hour into one bucket](#)

```
{
  "viz": "query_value",
  "requests": [
    {
      "q": "max:my_metric{*}.rollup(sum, 3600)",
      "aggregator": "sum",
      "conditional_formats": [
        {
          "comparator": ">",
          "palette": "white_on_red"
        },
        {
          "comparator": ">=",
          "palette": "white_on_yellow"
        },
        {
          "comparator": "<",
          "palette": "white_on_green"
        }
      ]
    }
  ]
}
```

```

    },
    "autoscale": true,
    "precision": 2
  }
}

```

Edit
Overview
Correlations
1th
Past 1 Hour

- Select your visualization
 

Timeseries
**Query Value**
Table
Heat Map
Scatter Plot
Distribution
Top List
Change
Host Map
Geomap
- Graph your data
 

Edit
JSON
Share
Custom Links

Metric: **my\_metric**
from: **everywhere**
max by: **everything**
rollup: **sum**
every: **1**
hours

Take the **Sum** value from the displayed timeframe.

**Units and formatting**

Show: **2 decimals**
Align:

☒ Autoformat (e.g. 1000000 → 1M)
☐ Use Custom Units

## The entire JSON

```

{
  "title": "Visualizing Data",
  "description": "## Visualizing Data\n\nIncludes: Custom metric over host, Metric on the DB, Custom metric with rollup function \n",
  "widgets": [
    ...
  ],
  "template_variables": [],
  "layout_type": "ordered",
  "is_read_only": false,
  "notify_list": [],
  "id": "66f-vy9-aqx"
}

```

## Create new timeboard API Call

Overview
GET Validate API key
POST Create a new da...
Datadog Authentication

Datadog API Collection / Dashboards / Create a new dashboard
Save

POST (baseurl)/api/v1/dashboard
Send

Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings

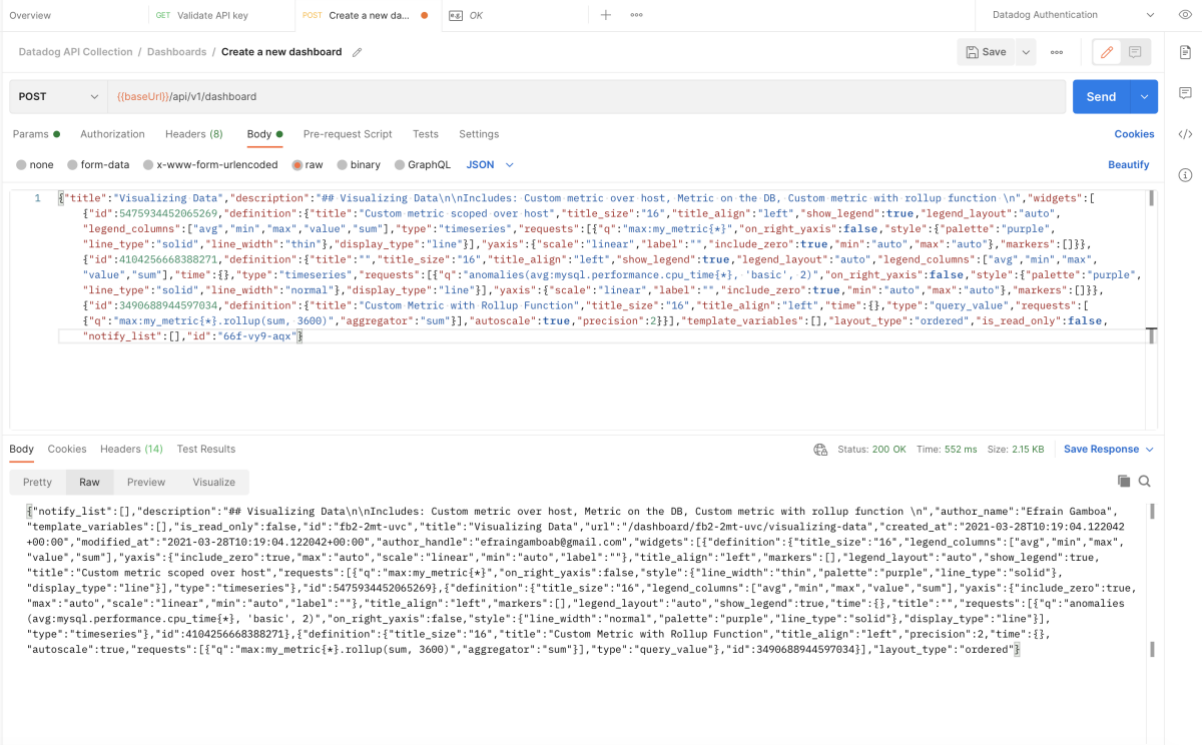
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON

1
{
 "title": "Visualizing Data",
 "description": "## Visualizing Data\n\nIncludes: Custom metric over host, Metric on the DB, Custom metric with rollup function \n",
 "widgets": [
 {
 "id": "5475934452065269",
 "definition": {
 "title": "Custom metric scoped over host",
 "title\_size": "16",
 "title\_align": "left",
 "show\_legend": true,
 "legend\_layout": "auto",
 "legend\_columns": ["avg", "min", "max", "value", "sum"],
 "type": "timeseries",
 "requests": [
 {
 "q": "max:my\_metric[\*]",
 "on\_right\_axis": false,
 "style": {
 "palette": "purple",
 "line\_type": "solid",
 "line\_width": "thin",
 "display\_type": "line"
 },
 "yaxis": {
 "scale": "linear",
 "label": "",
 "include\_zero": true,
 "min": "auto",
 "max": "auto",
 "markers": []
 }
 },
 {
 "id": "418425668388271",
 "definition": {
 "title": "",
 "title\_size": "16",
 "title\_align": "left",
 "show\_legend": true,
 "legend\_layout": "auto",
 "legend\_columns": ["avg", "min", "max", "value", "sum"],
 "type": "timeseries",
 "requests": [
 {
 "q": "anomalies(avg:mysql.performance.cpu\_time[\*], 'basic', 2)",
 "on\_right\_axis": false,
 "style": {
 "palette": "purple",
 "line\_type": "solid",
 "line\_width": "normal",
 "display\_type": "line"
 },
 "yaxis": {
 "scale": "linear",
 "label": "",
 "include\_zero": true,
 "min": "auto",
 "max": "auto",
 "markers": []
 }
 }
 ]
 }
 }
 ]
 },
 "id": "3490688944597034",
 "definition": {
 "title": "Custom Metric with Rollup Function",
 "title\_size": "16",
 "title\_align": "left",
 "time": {},
 "type": "query\_value",
 "requests": [
 {
 "q": "max:my\_metric[\*].rollup(sum, 3600)",
 "aggregator": "sum"
 }
 ],
 "autoscale": true,
 "precision": 2
 },
 "template\_variables": [],
 "layout\_type": "ordered",
 "is\_read\_only": false,
 "notify\_list": []
 },
 {
 "id": "66f-vy9-aqx"
 }
 ],
 "template\_variables": [],
 "layout\_type": "ordered",
 "is\_read\_only": false,
 "notify\_list": []
}

Response

Hit Send to get a response

## Create new timeboard response



Overview GET Validate API key POST Create a new da... OK + ... Datadog Authentication

Datadog API Collection / Dashboards / Create a new dashboard Save

POST {{baseUrl}}/api/v1/dashboard Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

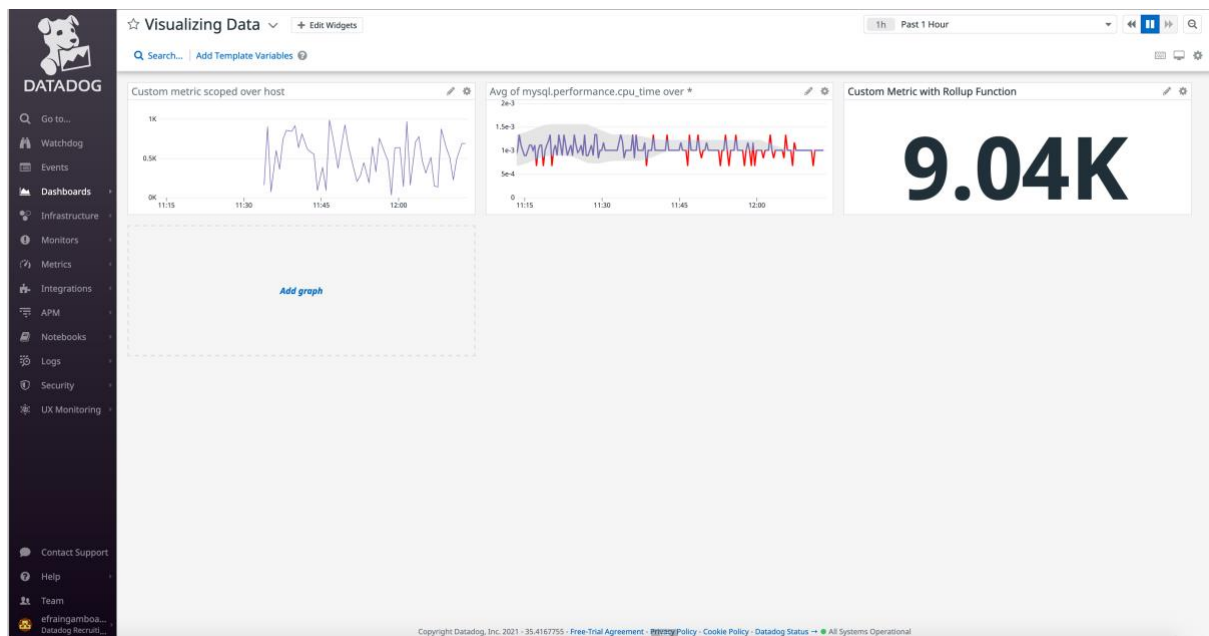
```
1 {
  "title": "Visualizing Data",
  "description": "## Visualizing Data\n\nIncludes: Custom metric over host, Metric on the DB, Custom metric with rollup function\n",
  "widgets": [
    {
      "id": "5475934452065269",
      "definition": {
        "title": "Custom metric scoped over host",
        "title_size": "16",
        "title_align": "left",
        "show_legend": true,
        "legend_layout": "auto",
        "legend_columns": ["avg", "min", "max", "value", "sum"],
        "type": "timeseries",
        "requests": [
          {
            "q": "max:my_metric[*]",
            "on_right_yaxis": false,
            "style": {
              "palette": "purple",
              "line_type": "solid",
              "line_width": "thin",
              "display_type": "line"
            },
            "yaxis": {
              "scale": "linear",
              "label": "",
              "include_zero": true,
              "min": "auto",
              "max": "auto",
              "markers": []
            }
          },
          {
            "id": "4104256668388271",
            "definition": {
              "title": "",
              "title_size": "16",
              "title_align": "left",
              "show_legend": true,
              "legend_layout": "auto",
              "legend_columns": ["avg", "min", "max", "value", "sum"],
              "time": {},
              "type": "timeseries",
              "requests": [
                {
                  "q": "anomalies(avg:mysql.performance.cpu.time[*], 'basic', 2)",
                  "on_right_yaxis": false,
                  "style": {
                    "palette": "purple",
                    "line_type": "solid",
                    "line_width": "normal",
                    "display_type": "line"
                  },
                  "yaxis": {
                    "scale": "linear",
                    "label": "",
                    "include_zero": true,
                    "min": "auto",
                    "max": "auto",
                    "markers": []
                  }
                }
              ]
            },
            "id": "3490688944597034",
            "definition": {
              "title": "Custom Metric with Rollup Function",
              "title_size": "16",
              "title_align": "left",
              "time": {},
              "type": "query_value",
              "requests": [
                {
                  "q": "max:my_metric[*].rollup(sum, 3600)",
                  "aggregator": "sum",
                  "autoscale": true,
                  "precision": 2
                }
              ],
              "template_variables": [],
              "layout_type": "ordered",
              "is_read_only": false,
              "notify_list": [],
              "id": "66f-vy9-aqx"
            }
          }
        ]
      },
      "type": "timeseries",
      "requests": [
        {
          "q": "max:my_metric[*]",
          "on_right_yaxis": false,
          "style": {
            "palette": "purple",
            "line_type": "solid",
            "line_width": "thin",
            "display_type": "line"
          },
          "yaxis": {
            "scale": "linear",
            "label": "",
            "include_zero": true,
            "min": "auto",
            "max": "auto",
            "markers": []
          }
        },
        {
          "id": "4104256668388271",
          "definition": {
            "title": "",
            "title_size": "16",
            "title_align": "left",
            "show_legend": true,
            "legend_layout": "auto",
            "legend_columns": ["avg", "min", "max", "value", "sum"],
            "time": {},
            "type": "timeseries",
            "requests": [
              {
                "q": "anomalies(avg:mysql.performance.cpu.time[*], 'basic', 2)",
                "on_right_yaxis": false,
                "style": {
                  "palette": "purple",
                  "line_type": "solid",
                  "line_width": "normal",
                  "display_type": "line"
                },
                "yaxis": {
                  "scale": "linear",
                  "label": "",
                  "include_zero": true,
                  "min": "auto",
                  "max": "auto",
                  "markers": []
                }
              }
            ]
          },
          "id": "3490688944597034",
          "definition": {
            "title": "Custom Metric with Rollup Function",
            "title_size": "16",
            "title_align": "left",
            "time": {},
            "type": "query_value",
            "requests": [
              {
                "q": "max:my_metric[*].rollup(sum, 3600)",
                "aggregator": "sum",
                "autoscale": true,
                "precision": 2
              }
            ],
            "template_variables": [],
            "layout_type": "ordered",
            "is_read_only": false,
            "notify_list": [],
            "id": "66f-vy9-aqx"
          }
        }
      ]
    }
  ]
}
```

Body Cookies Headers (14) Test Results Status: 200 OK Time: 552 ms Size: 2.15 KB Save Response

Pretty Raw Preview Visualize

```
{
  "notify_list": [],
  "description": "## Visualizing Data\n\nIncludes: Custom metric over host, Metric on the DB, Custom metric with rollup function\n",
  "author_name": "Efrain Gamboa",
  "template_variables": [],
  "is_read_only": false,
  "id": "fb2-2mt-uv",
  "title": "Visualizing Data",
  "url": "/dashboard/fb2-2mt-uv/visualizing-data",
  "created_at": "2021-03-28T10:19:04.122042+00:00",
  "modified_at": "2021-03-28T10:19:04.122042+00:00",
  "author_handle": "efraingamboa@gmail.com",
  "widgets": [
    {
      "definition": {
        "title_size": "16",
        "legend_columns": ["avg", "min", "max", "value", "sum"],
        "yaxis": {
          "include_zero": true,
          "max": "auto",
          "scale": "linear",
          "min": "auto",
          "label": ""
        },
        "title_align": "left",
        "markers": [],
        "legend_layout": "auto",
        "show_legend": true,
        "display_type": "line",
        "type": "timeseries",
        "id": "5475934452065269",
        "definition": {
          "title_size": "16",
          "legend_columns": ["avg", "min", "max", "value", "sum"],
          "yaxis": {
            "include_zero": true,
            "max": "auto",
            "scale": "linear",
            "min": "auto",
            "label": ""
          },
          "title_align": "left",
          "markers": [],
          "legend_layout": "auto",
          "show_legend": true,
          "time": {},
          "requests": [
            {
              "q": "anomalies(avg:mysql.performance.cpu.time[*], 'basic', 2)",
              "on_right_yaxis": false,
              "style": {
                "line_width": "normal",
                "palette": "purple",
                "line_type": "solid",
                "display_type": "line"
              }
            }
          ]
        },
        "type": "timeseries",
        "id": "4104256668388271",
        "definition": {
          "title_size": "16",
          "title": "Custom Metric with Rollup Function",
          "title_align": "left",
          "precision": 2,
          "time": {},
          "autoscale": true,
          "requests": [
            {
              "q": "max:my_metric[*].rollup(sum, 3600)",
              "aggregator": "sum",
              "type": "query_value",
              "id": "3490688944597034",
              "layout_type": "ordered"
            }
          ]
        }
      }
    }
  ]
}
```

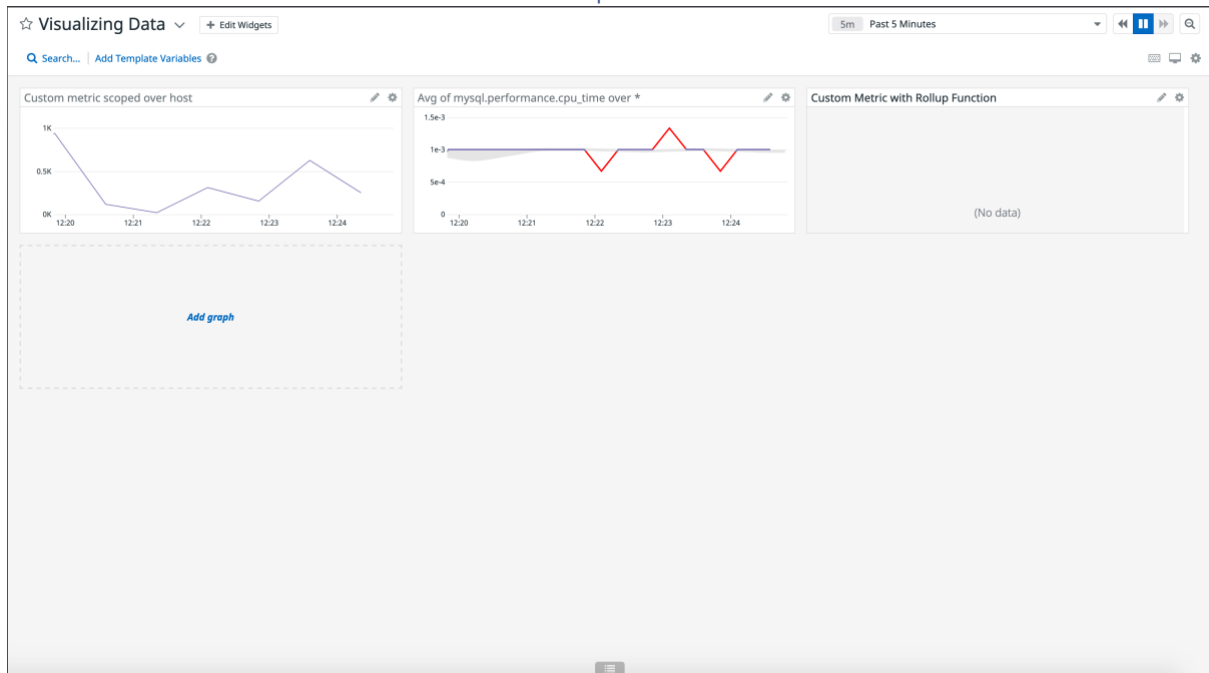
Then verified that the timeboard was created by checking in the UI



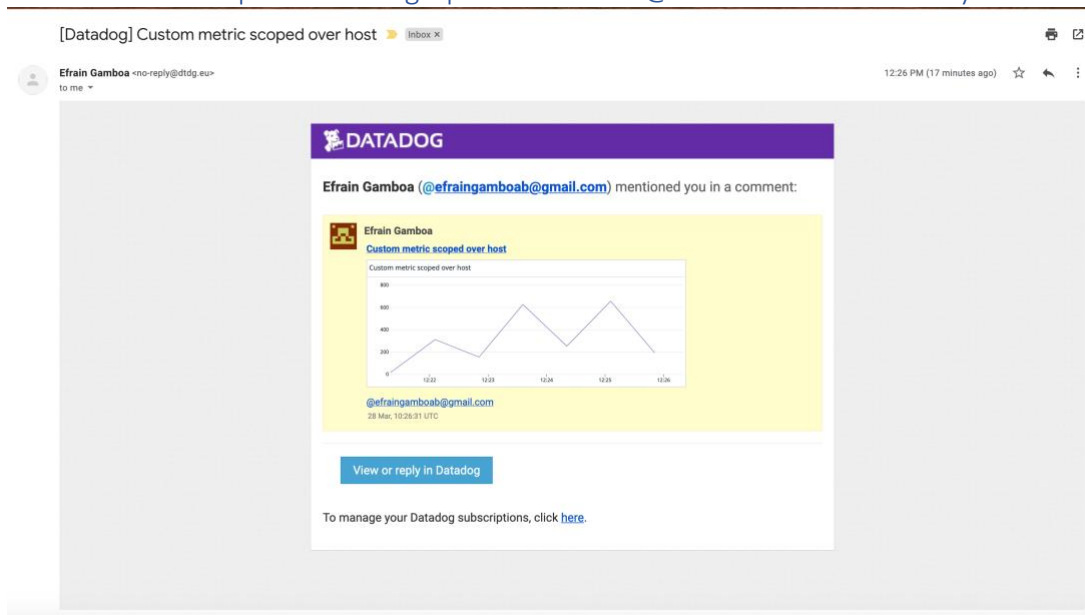
Dashboard public url: <https://p.datadoghq.eu/sb/478y64jbxcdcxv9i-f0c844c10bcbe0b3922a5e33da046a3b>

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

- Set the Timeboard's timeframe to the past 5 minutes



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



Bonus Question: What is the Anomaly graph displaying?

From the datadog Anomaly function definition:

Anomaly detection is an algorithmic feature that identifies when a metric is behaving differently than it has in the past

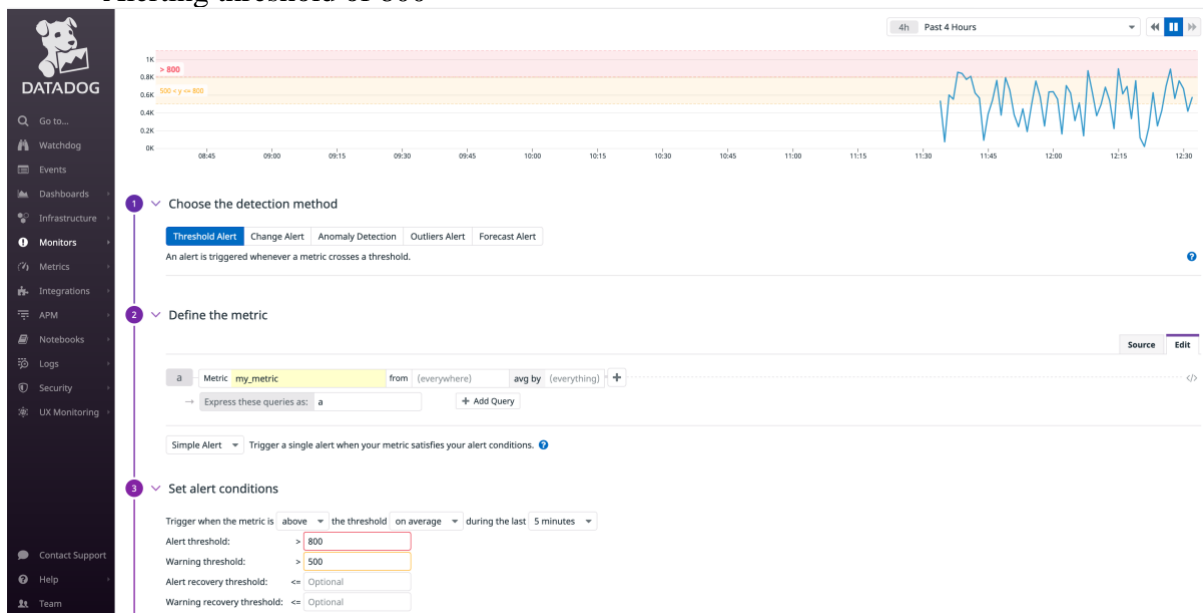
**Anomaly detection** - With the default option (`above` or `below`) a metric is considered to be anomalous if it is outside of the gray anomaly band. Optionally, you can specify whether being only `above` or `below` the bands is considered anomalous.

## Monitoring Data

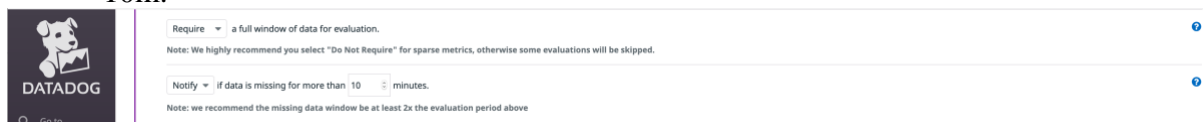
Since you've already caught your test metric going above 800 once, you don't want to have to continually watch this dashboard to be alerted when it goes above 800 again. So let's make life easier by creating a monitor.

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



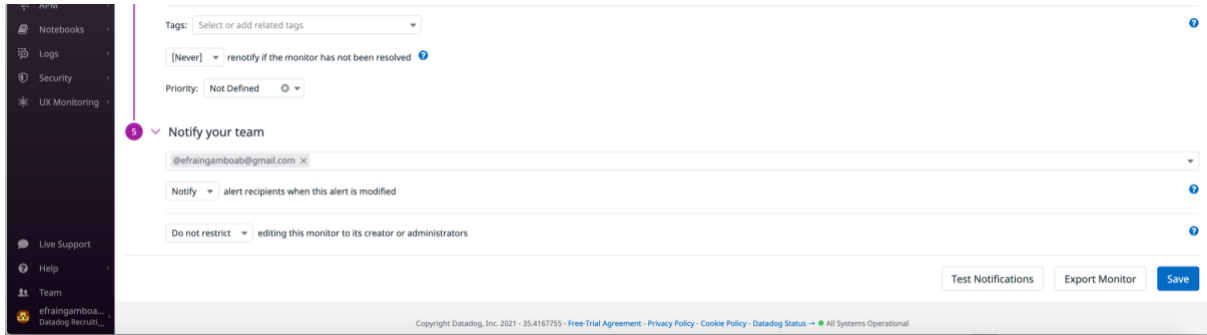
- And also ensure that it will notify you if there is No Data for this query over the past 10m.




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.



```

{{#is_alert}}Alert! my_metric is over 800: value {{value}} with host_ip:
{{host_ip}} {{/is_alert}}
{{#is_warning}}Warning! my_metric is over 500: value {{value}} {{/is_warning}}
{{#is_no_data}}No data received {{/is_no_data}}
{{#is_recovery}}my_metrics is OK {{/is_recovery}}
@efraingamboab@gmail.com

```

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

## Alert

The alert wasn't triggering, so I changed the random values to be higher so the average would be over 800.

```
import random

from datadog_checks.base import AgentCheck

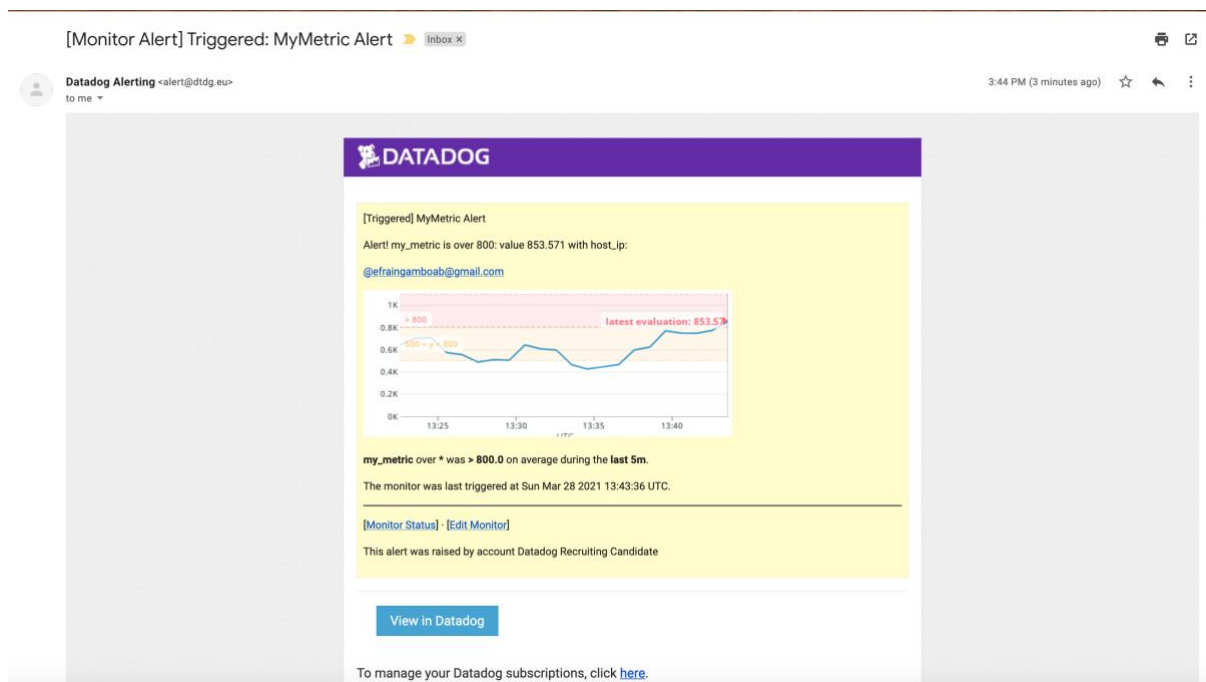
__version__ = "1.0.0"

class MyClass(AgentCheck):
    def check(self, instance):

        # Calling the functions below twice simulates
        # several metrics submissions during one Agent run.
        self.gauge(
            "my_metric",
            random.randint(500, 1000),
            tags=["env:dev", "metric_submission_type:gauge"],
        )
        self.gauge(
            "my_metric",
            random.randint(500, 1000),
            tags=["env:dev", "metric_submission_type:gauge"],
        )

"my_metric.py" 22L, 571C 19,29 All
```

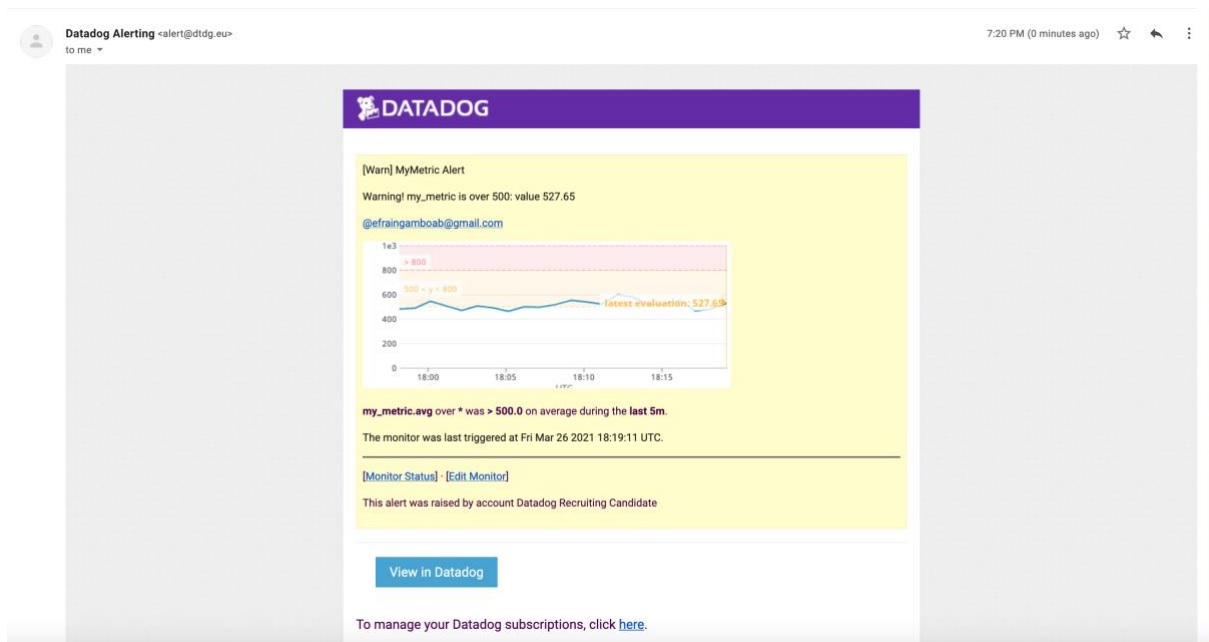
After making this change the alert was finally triggered



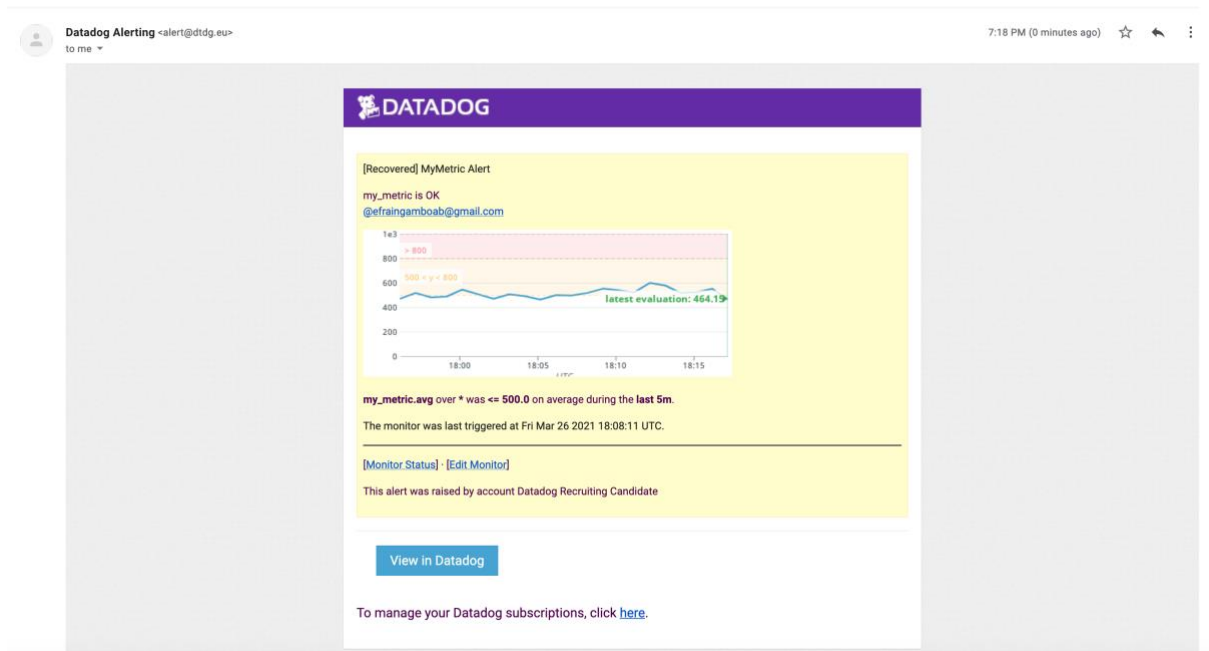
Looking at the message received it is not displaying the host\_ip. From what I understand reading the documentations it takes it from the agent config file in the tags. Since I haven't specified it there it's not displaying

Testing with the `{{host.name}}` instead, but it didn't work either.

## Warn




## Recovered



## No data received

[Monitor Alert] No data: MyMetric Alert Trash X

Datadog Alerting <alert@datadg.eu> to me 5:06 PM (0 minutes ago)

 **DATADOG**

[No data] MyMetric Alert

No data received

[@efraingamboa@gmail.com](mailto:efraingamboa@gmail.com)

**The monitor has been missing data for the last 10m**

The monitor was last triggered at Sun Mar 28 2021 15:05:36 UTC.

[\[Monitor Status\]](#) [\[Edit Monitor\]](#)

This alert was raised by account Datadog Recruiting Candidate

[View in Datadog](#)

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

Reply Forward

**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

In order to do this it's necessary to schedule downtime.

The process is quite straight forward:

- Navigate to "Monitor" -> "Manage Downtime"
- Select the "manage downtime" tab
- Click on "Schedule downtime"

**2** Schedule

One Time **Recurring**

Start Date:  Time Zone:

Repeat Every:

Repeat On: ☐ Sun ☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☐ Sat

Beginning:

Duration:

Repeat Until:

Summary: **From 7:00pm to 9:00am next day**  
**Weekly on Monday, Tuesday, Wednesday, Thursday, and Friday**

[Preview planned recurrences](#)

One that silences it all day on Sat-Sun


**2**

### Schedule


One Time **Recurring**

Start Date:	<input type="text" value="2021/03/27"/>	Time Zone:	<input type="text" value="Europe/Madrid"/>
Repeat Every:	<input type="text" value="1"/> <input type="text" value="weeks"/>		
Repeat On:	<input checked="" type="checkbox"/> Sun <input type="checkbox"/> Mon <input type="checkbox"/> Tue <input type="checkbox"/> Wed <input type="checkbox"/> Thu <input type="checkbox"/> Fri <input checked="" type="checkbox"/> Sat		
Beginning:	<input type="text" value="00:00"/>		
Duration:	<input type="text" value="1"/> <input type="text" value="days"/>		
Repeat Until:	<input type="text" value="No end date"/>		
Summary:	From 12:00am to 12:00am next day Weekly on Sunday and Saturday		
<a href="#">Preview planned recurrences</a>			

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

 **DATADOG**

**A Datadog event** mentioned you:

**Scheduled downtime on MyMetric Alert started**  
Scheduled downtime on [MyMetric Alert](#) has started.  
Alerting on [MyMetric Alert](#) will be silenced until 8:30AM UTC on March 27.  
[@efraingamboab@gmail.com](mailto:@efraingamboab@gmail.com)  
26 Mar, 18:30:08 UTC

[View or reply in Datadog](#)

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

## Collecting APM Data

Deploy the Flask app with Python

```
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(__name__)
```

```
@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(host='0.0.0.0', port='5050')
```

## Initial Setup

In order to deploy the given Flask App in my EC2 instance there are some necessary previous steps to take to have the environment ready.

These are:

### Install virtualenv

```
$ sudo apt-get install python-virtualenv
$ mkdir datadogflask
$ cd datadogflask
$ sudo apt-get install python3-venv
$ python3 -m venv venv
$ . venv/bin/activate
$ pip install Flask
```

### Create the python file “datadogflaskapp.py”

```
In the virtual environment
(venv) ubuntu@ip-172-31-41-67:~/datadogflask$ export
FLASK_APP=datadogflaskapp.py
(venv) ubuntu@ip-172-31-41-67:~/datadogflask$ flask run --port 5050
```

```
Downloads — ubuntu@ip-172-31-41-67: ~/datadogflask — ssh -i...
*** System restart required ***
Last login: Mon Mar 29 07:54:18 2021 from 95.169.229.193
[ubuntu@ip-172-31-41-67:~]$ ls
datadogflask  ddagent-install.log  primer-dataset.json
[ubuntu@ip-172-31-41-67:~]$ cd datadogflask/
[ubuntu@ip-172-31-41-67:~/datadogflask]$ . venv/bin/activate
(env) ubuntu@ip-172-31-41-67:~/datadogflask$ export FLASK_APP=datadogflaskapp.py
(env) ubuntu@ip-172-31-41-67:~/datadogflask$ flask run --port 5050
 * Serving Flask app "datadogflaskapp.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2021-03-29 08:59:32,440 - werkzeug - INFO - * Running on http://127.0.0.1:5050/
(Press CTRL+C to quit)
```

And the datadogflaskapp is running.  
Now it's time to implement APM

## APM Setup

In the dashboard navigate to "APM"  
Then in the getting started section select "Host Based". And follow the instructions

The agent in this case is already installed.

Select the language. In this case python

In the virtual environment

```
(venv) ubuntu@ip-172-31-41-67:~/datadogflask$ DD_SERVICE="datadogflaskapp" DD_ENV="prod" DD_LOGS_INJECTION=true ddtrace-run python datadogflaskapp.py
```

```
Downloads — ubuntu@ip-172-31-41-67: ~/datadogflask — ssh -i...
[AC(venv) ubuntu@ip-172-31-41-67:~/datadogflask$ DD_SERVICE="datadogflaskapp" DD_ENV="prod" DD_LOGS_INJECTION=true ddtrace-run python datadogflaskapp.py
 * Serving Flask app "datadogflaskapp" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2021-03-29 09:01:28,302 INFO [werkzeug] [_internal.py:113] [dd.service=datadogflaskapp dd.env=prod dd.version= dd.trace_id=0 dd.span_id=0] - * Running on http://0.0.0.0:5050/ (Press CTRL+C to quit)
2021-03-29 09:01:28,302 - werkzeug - INFO - * Running on http://0.0.0.0:5050/ (Press CTRL+C to quit)
```

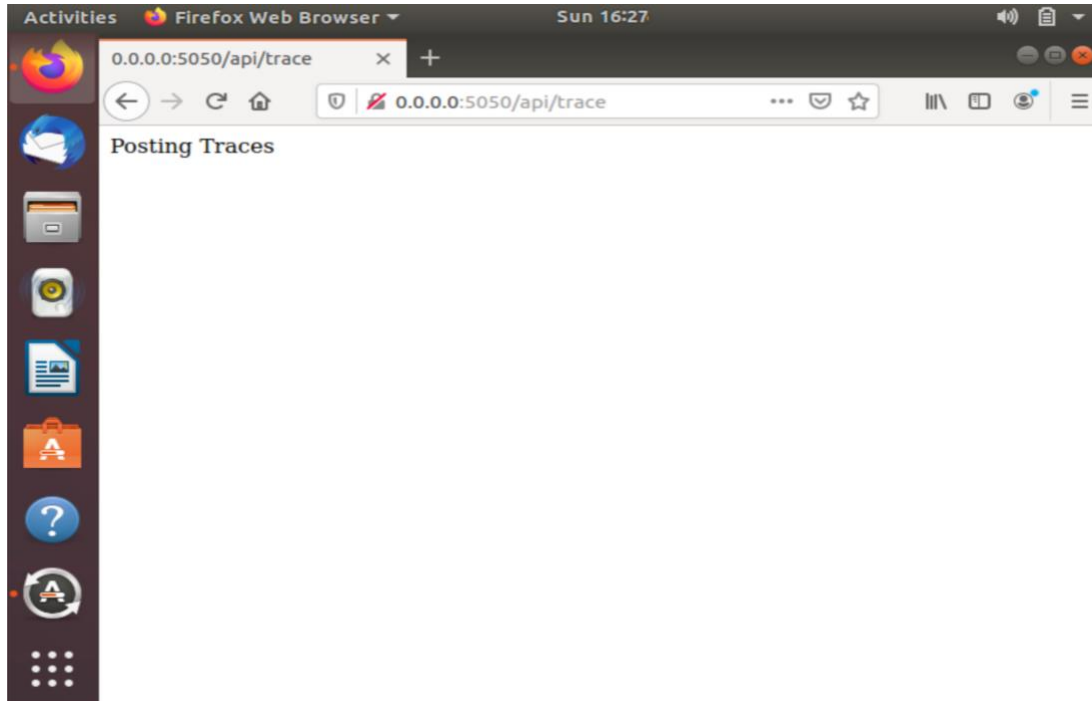
Now the application is running.

Since I had already install the GUI for my ubuntu machine I went to the browser and navigated to the url:

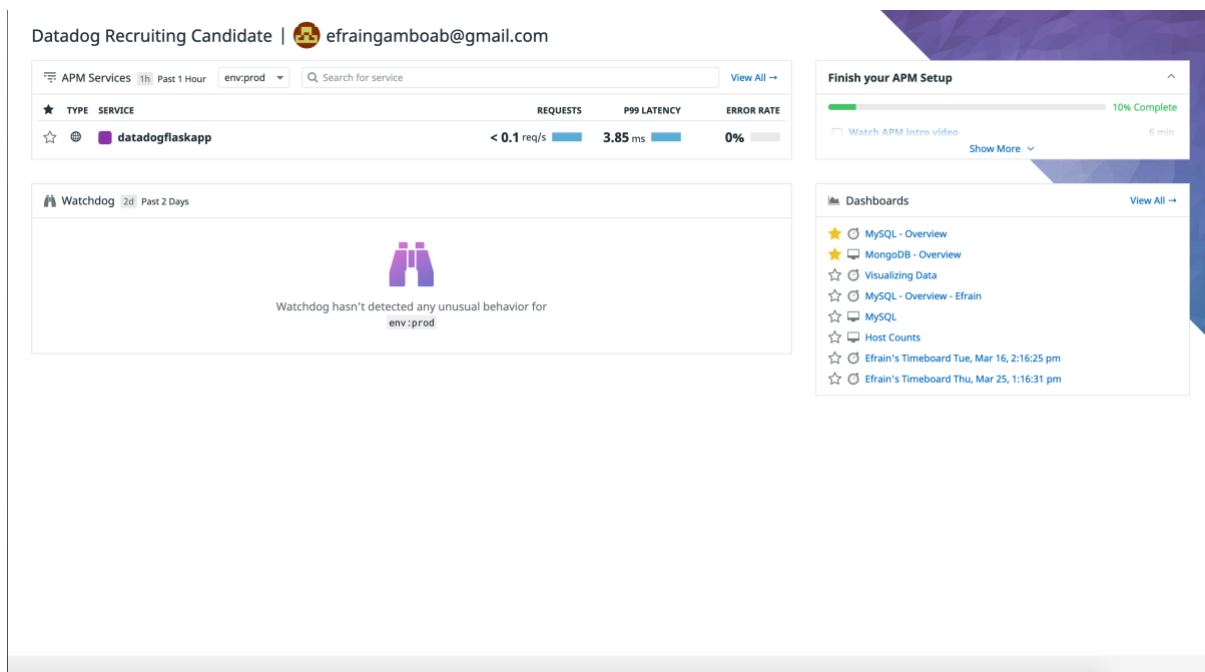
<http://0.0.0.0:5050/>

<http://0.0.0.0:5050/api/apm>

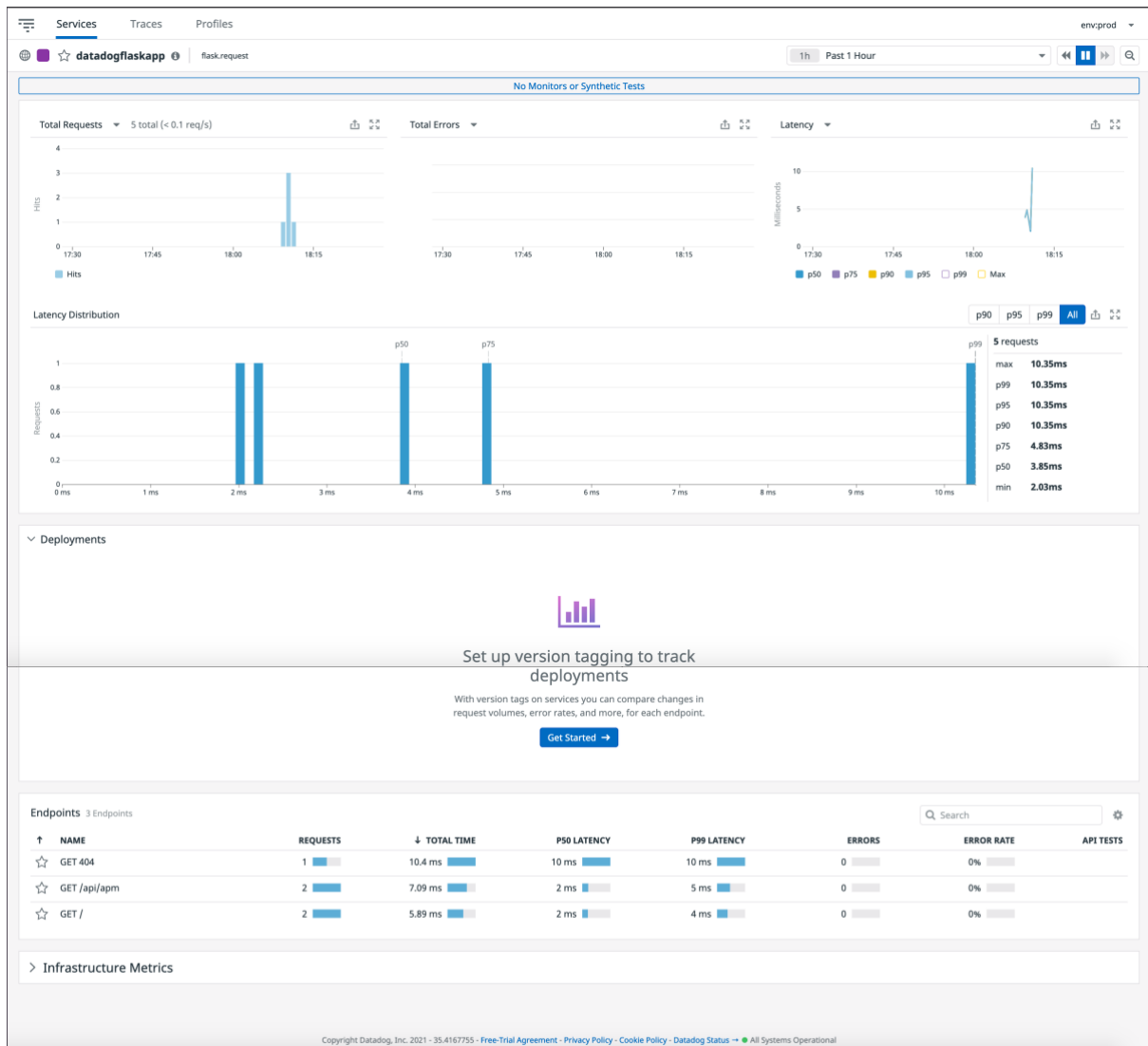
<http://0.0.0.0:5050/api/trace>

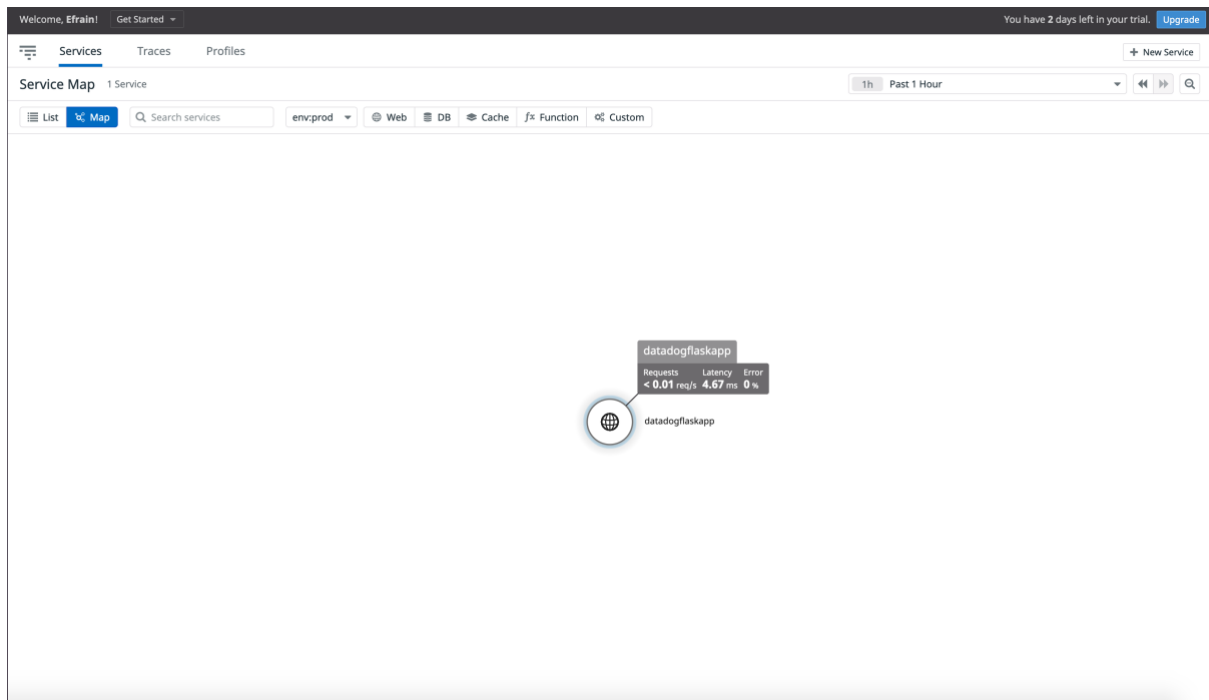


Now in the datadog UI. It is possible to see the below screen under “APM”. And the datadogflasapp listed and reporting data.









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

- Services group together endpoints, queries, or jobs for the purpose of scaling instances. All services are listed under **Service List** and can be visually seen as in a Micro-service format under **Service Map**.
- Resources represent a particular domain of a Customer Application. They could typically be an instrumented web endpoint, database query, or background job. Each resource has its own Resource page with trace metrics scoped to the specific endpoint.

## Final Question

Is there anything creative you would use Datadog for?

I live in Barcelona where there currently is a big initiative for a greener city. So one use case that comes to mind is tracking the number cars in the low emissions area.

There are cameras checking the license plates of all vehicles entering that area. If this was publicly available can be monitored.

Another idea would be during the winter season check the mountain and slopes snow site to know if it's good to go skiing. Also compare it with previous year's data and use it to determine global warming rates.

## Extra Information

### Mysql DB

I follow this guide: <https://www.digitalocean.com/community/tutorials/a-basic-mysql-tutorial>

Mysql datadog user:

```
CREATE USER 'datadog'@'localhost' IDENTIFIED BY 'datadog1!'
```

### Mongo DB

Simultaneously I tried to implement also the mongodb integration. So I set up a mongo DB deployment following this guide:

<https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-18-04-source>

Then imported a DB

Following the steps explained in this guide:

<https://www.digitalocean.com/community/tutorials/how-to-import-and-export-a-mongodb-database-on-ubuntu-20-04>

And then followed the integration steps

```
db.createUser({
  "user": "datadog",
  "pwd": "datadog1!",
  "roles": [
    { role: "read", db: "admin" },
    { role: "clusterMonitor", db: "admin" },
    { role: "read", db: "local" }
  ]
})
```



```
vagrant — vagrant@vagrant: /etc/datadog-agent/conf.d/mongo.d...
mongo (2.2.0)
-----

instance 0:

could not invoke 'mongo' python check constructor. New constructor API r
etuned:
Traceback (most recent call last):
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/datadog_checks/m
ongo/mongo.py", line 70, in __init__
    self.config = MongoConfig(self.instance, self.log)
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/datadog_checks/m
ongo/config.py", line 48, in __init__
    self.clean_server_name = self._get_clean_server_name()
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/datadog_checks/m
ongo/config.py", line 84, in _get_clean_server_name
    return parse_mongo_uri(server, sanitize_username=bool(self.ssl_params))[4]
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/datadog_checks/m
ongo/utils.py", line 41, in parse_mongo_uri
    parsed = pymongo.uri_parser.parse_uri(server)
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/pymongo/uri_pars
er.py", line 391, in parse_uri
    nodes = split_hosts(hosts, default_port=default_port)
  File "/opt/datadog-agent/embedded/lib/python3.8/site-packages/pymongo/uri_pars
```

## Custom metric

In order to understand how the custom metric works and the what the different functions do I created the metric exactly as it was in the example.

```
vagrant — vagrant@vagrant: /etc/datadog-agent/checks.d — ssh • vagrant ssh...
import random

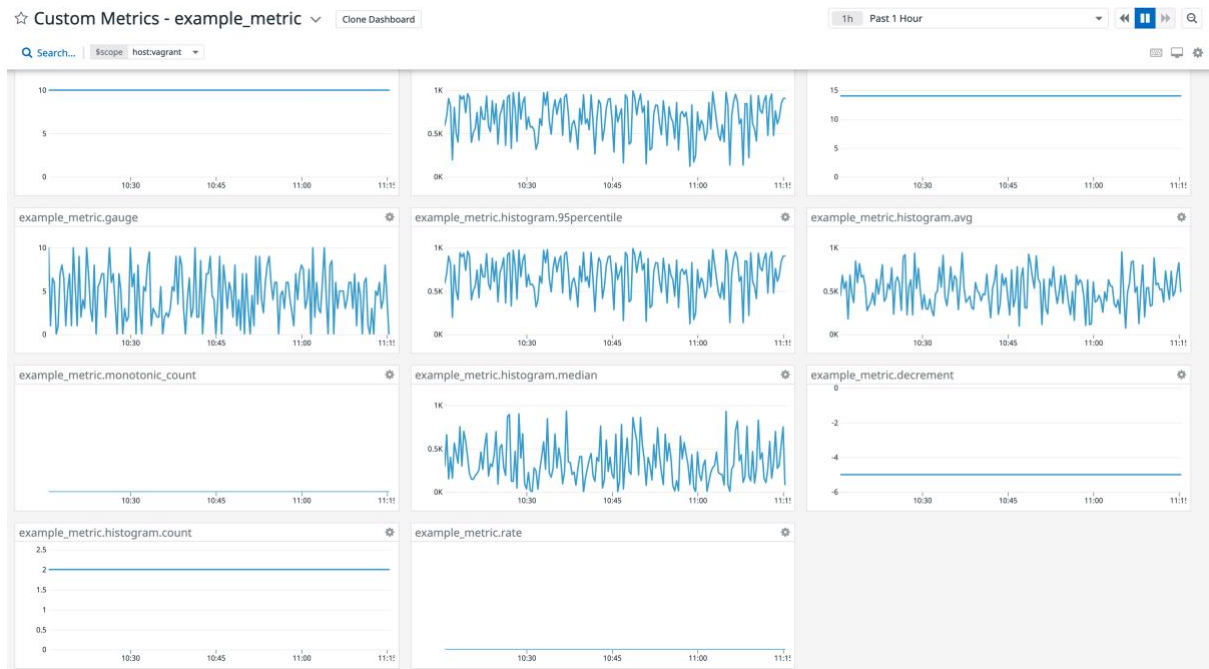
from datadog_checks.base import AgentCheck

__version__ = "1.0.0"

class MyClass(AgentCheck):
    def check(self, instance):
        self.count(
            "example_metric.count",
            10,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.count(
            "example_metric.decrement",
            -5,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.count(
            "example_metric.increment",
            14,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.rate(
            "example_metric.rate",
            3,
            tags=["env:dev", "metric_submission_type:rate"],
        )
        self.gauge(
            "example_metric.gauge",
            random.randint(0, 10),
            tags=["env:dev", "metric_submission_type:gauge"],
        )
        self.monotonic_count(
            "example_metric.monotonic_count",
            2,
            tags=["env:dev", "metric_submission_type:monotonic_count"],
        )

        # Calling the functions below twice simulates
        # several metrics submissions during one Agent run.
        self.histogram(
            "example_metric.histogram",
            random.randint(0, 1000),
            tags=["env:dev", "metric_submission_type:histogram"],
        )
        self.histogram(
            "example_metric.histogram",
            random.randint(0, 1000),
            tags=["env:dev", "metric_submission_type:histogram"],
        )
```

And obtained the following result.



```
import random
```

```
from datadog_checks.base import AgentCheck
```

```
__version__ = "1.0.0"
```

```
class MyClass(AgentCheck):
    def check(self, instance):
        self.count(
            "example_metric.count",
            10,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.count(
            "example_metric.decrement",
            -5,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.count(
            "example_metric.increment",
            14,
            tags=["env:dev", "metric_submission_type:count"],
        )
        self.rate(
            "example_metric.rate",
            3,
            tags=["env:dev", "metric_submission_type:rate"],
        )
        self.gauge(
            "example_metric.gauge",
            random.randint(0, 10),
            tags=["env:dev", "metric_submission_type:gauge"],
        )
        self.monotonic_count(
```

```
"example_metric.monotonic_count",
2,
tags=["env:dev","metric_submission_type:monotonic_count"],
)

# Calling the functions below twice simulates
# several metrics submissions during one Agent run.
self.histogram(
    "example_metric.histogram",
    random.randint(0, 1000),
    tags=["env:dev","metric_submission_type:histogram"],
)
self.histogram(
    "example_metric.histogram",
    random.randint(0, 1000),
    tags=["env:dev","metric_submission_type:histogram"],
)

import random

from datadog_checks.base import AgentCheck

__version__ = "1.0.0"

class MyClass(AgentCheck):
    def check(self, instance):
        self.gauge(
            "my_metric",
            random.randint(0, 1000),
            tags=["env:dev","metric_submission_type:gauge"],

gauge()
```

This function submits the value of a metric at a given timestamp. If called multiple times during a check's execution for a metric only the last sample is used

`histogram()`

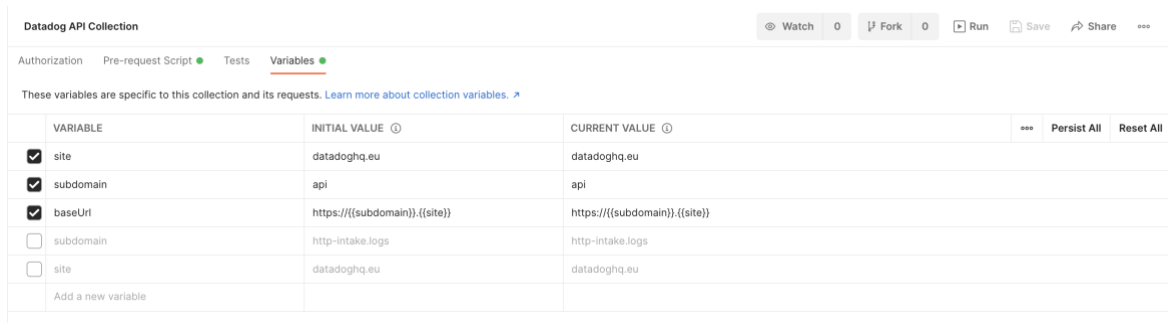
This function submits the sample of a histogram metric that occurred during the check interval. It can be called multiple times during a check's execution, each sample being added to the statistical distribution of the set of values for this metric.

Then I settled for the gauge option.

## Visualizing Data

I encountered difficulties using postman to interact with the API. Specifically around authentication.

I changed the environment variables with the same API key I used for the integrations, and it wasn't working. Also checked that the URL was the correct one (datadoghq.eu).



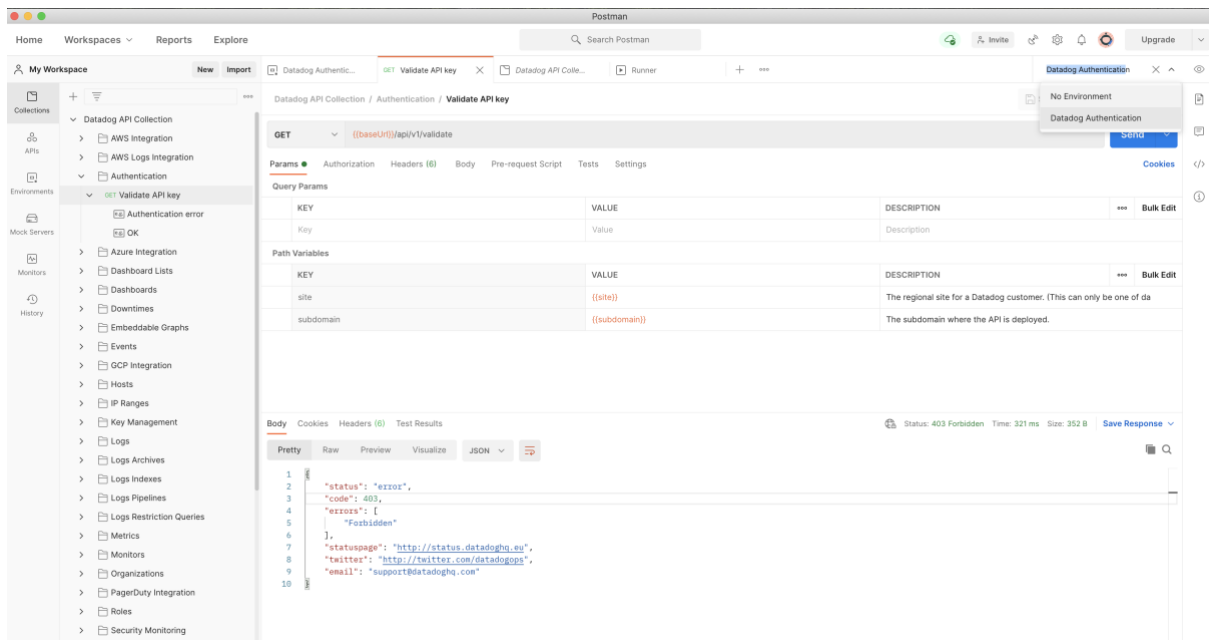
Datadog API Collection

Authorization Pre-request Script Tests Variables

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE	CURRENT VALUE		Persist All	Reset All
<input checked="" type="checkbox"/>	site	datadoghq.eu	datadoghq.eu			
<input checked="" type="checkbox"/>	subdomain	api	api			
<input checked="" type="checkbox"/>	baseUrl	https://{{subdomain}}.{{site}}	https://{{subdomain}}.{{site}}			
<input type="checkbox"/>	subdomain	http-intake.logs	http-intake.logs			
<input type="checkbox"/>	site	datadoghq.eu	datadoghq.eu			
	Add a new variable					

And it was always returning code 403.



Postman

Home Workspaces Reports Explore

My Workspace

Datadog API Collection / Authentication / Validate API key

GET {{baseUrl}}/api/v1/validate

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
site	{{site}}	The regional site for a Datadog customer. (This can only be one of da
subdomain	{{subdomain}}	The subdomain where the API is deployed.

Body Cookies Headers (6) Test Results

Status: 403 Forbidden Time: 321 ms Size: 352 B Save Response

```
1 {
2   "status": "error",
3   "code": 403,
4   "errors": [
5     "Forbidden"
6   ],
7   "statuspage": "http://status.datadoghq.eu",
8   "twitter": "http://twitter.com/datadogops",
9   "email": "support@datadoghq.com"
10 }
```

I generated a new API key but that didn't work either. The reasoning behind this was that I believed that I thought that since the key was being used in the integrations it might've had an impact on using the same key in postman.

I tried using the web version of postman and it started working.