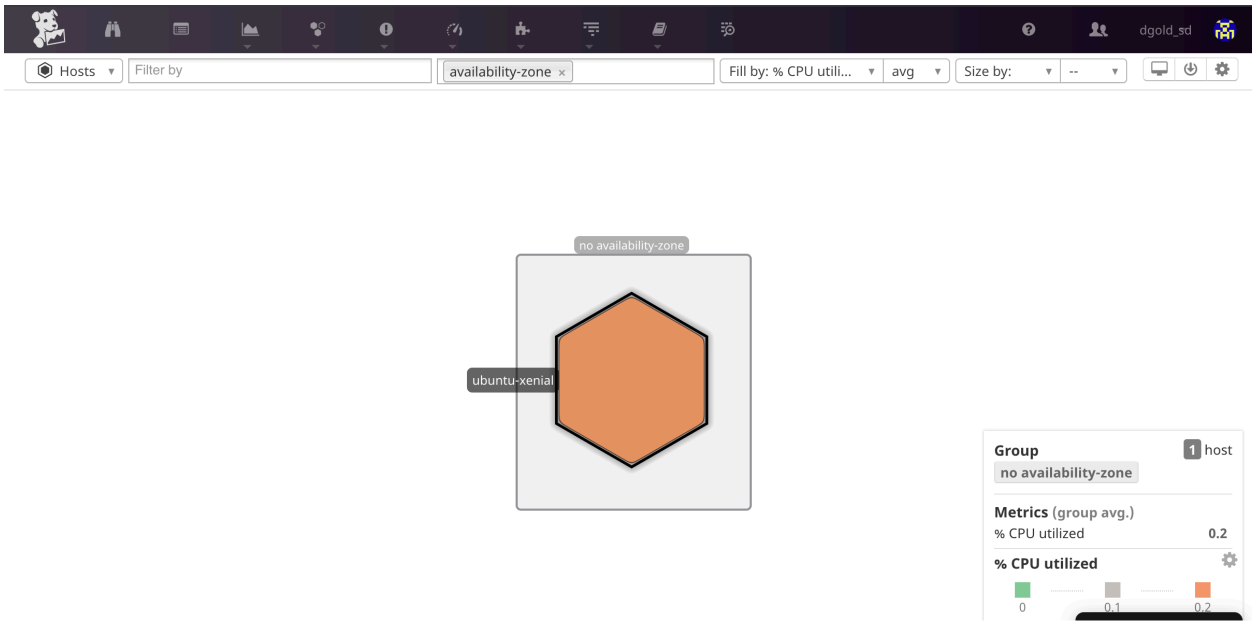
Getting started with Datadog is extremely easy. The product is provided as Software-as-a-Service (SaaS) so there is nothing to install and configure. Agents are installed locally to collect data and send it to Datadog. Agents are easily installed to monitor servers, applications, and services whether in your datacenter or public cloud. To start, an ubuntu server was deployed and the agent was installed, using the command:

```
DD_API_KEY=48ff5cf94558d3471524bf7d894050b8 bash -c "$(curl -L
https://raw.githubusercontent.com/DataDog/datadog-agent/master/cmd/
agent/install_script.sh)"
```

This not only did the installation but also started the agent. From there, the host became visible in the UI.



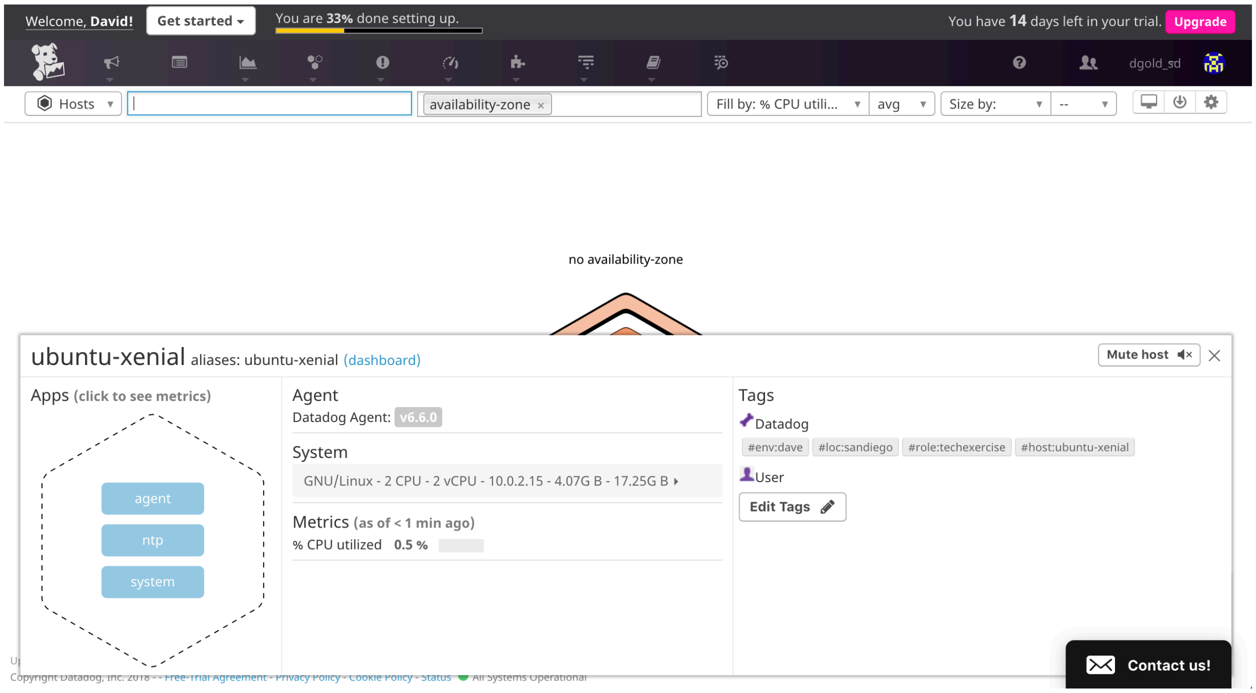
Group 1 host
no availability-zone
Metrics (group avg.)
% CPU utilized 0.2
% CPU utilized
0 0.1 0.2

Tags were added in the Agent config file. Tags allow metrics to be filtered, aggregated, and compared. There are different ways to add tags. Here, the tags were added to the agent configuration file on the ubuntu server:

In /etc/datadog-agent/datadog.yaml, the following line was added:

```
# Set the host's tags (optional)
tags: loc:SanDiego, env:dave, role:techExercise
```

These tags can then be seen in the UI, as shown here on the host maps page:



ubuntu-xenial aliases: ubuntu-xenial (dashboard)

Apps (click to see metrics): agent, ntp, system

Agent
Datadog Agent: v6.6.0

System
GNU/Linux - 2 CPU - 2 vCPU - 10.0.2.15 - 4.07G B - 17.25G B

Metrics (as of < 1 min ago)
% CPU utilized 0.5 %

Tags
Datadog
#env:dave #loc:sandiego #role:techexercise #host:ubuntu-xenial
User
Edit Tags

There are over 200 built-in integrations. For this exercise, PostgreSQL was installed on the ubuntu server and a simple database was configured.:

```
> apt-get install postgresql postgresql-contrib
```

The Datadog PostgreSQL integration is well documented and it was easy to install following the instructions at

https://docs.datadoghq.com/integrations/postgres/#installation

The following changes were made to configure the metrics collection:

In /etc/datadog-agent/conf.d/postgres.d/conf.yaml

```
instances:
  - host: localhost
```

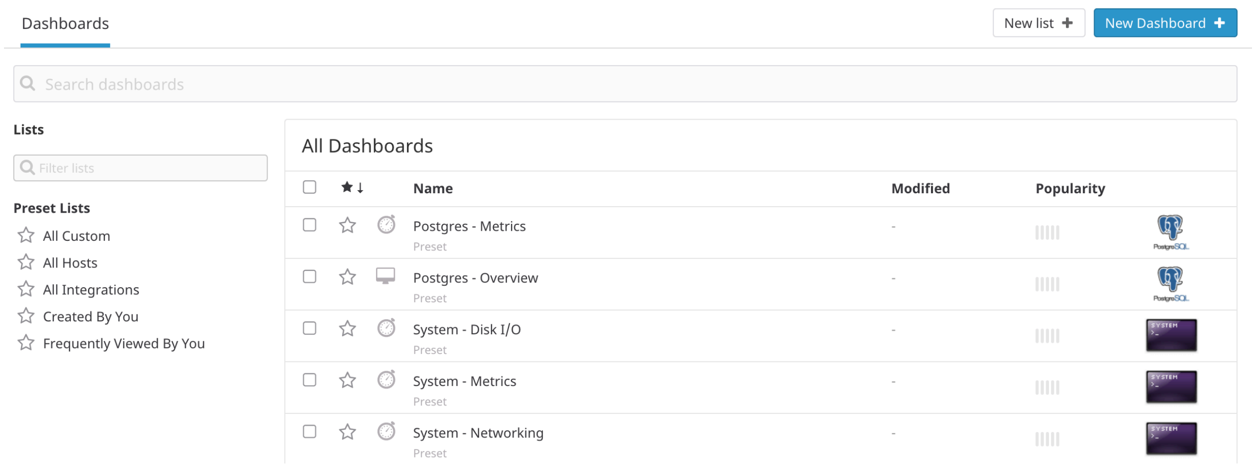
```
    port: 5432
    username: datadog
    password: datadog
    dbname: Dave

…

## log Section (Available for Agent >=6.0)
logs:

  - type : file
    path : /var/lib/postgresql/9.5/main/pg_log/pg.log
    service : postgres
    source : postgresql
    sourcecategory : database
    #   tags: (optional) add tags to each logs collected
```

To enable collection of data from the PostgreSQL logs, the following changes were made:

In /etc/postgresql/9.5/main/postgresql.conf

```
# This is used when logging to stderr:
logging_collector = on          # Enable capturing of stderr and
csvlog
                                        # into log files. Required to
be on for
                                        # csvlogs.
                                        # (change requires restart)

# These are only used if logging_collector is on:
log_directory = 'pg_log'                # directory where log files
are written,
                                        # can be absolute or relative
to PGDATA
log_filename = 'pg.log' # log file name pattern,
                                        # can include strftime()
escapes
log_file_mode = 0644                    # creation mode for log files,
                                        # begin with 0 to use octal
notation
log_line_prefix = '%m [%p] %d %a %u %h %c '                     #
special values:
log_statement = 'all'                   # none, ddl, mod, all
```

In the UI, the PostgreSQL data can then be seen in two different dashboards, ‘Postgres - Metrics’ and ‘Postgres - Overview’:



All Dashboards

Name	Modified	Popularity
Postgres - Metrics Preset	-	
Postgres - Overview Preset	-	
System - Disk I/O Preset	-	
System - Metrics Preset	-	
System - Networking Preset	-	

While there are over 200 built-in metrics, there will always be situations were custom metrics are required. This may be because of custom applications or unique systems. Here, a custom Agent check will be created that submits a metric named my_metric with a random value between 0 and 1000.

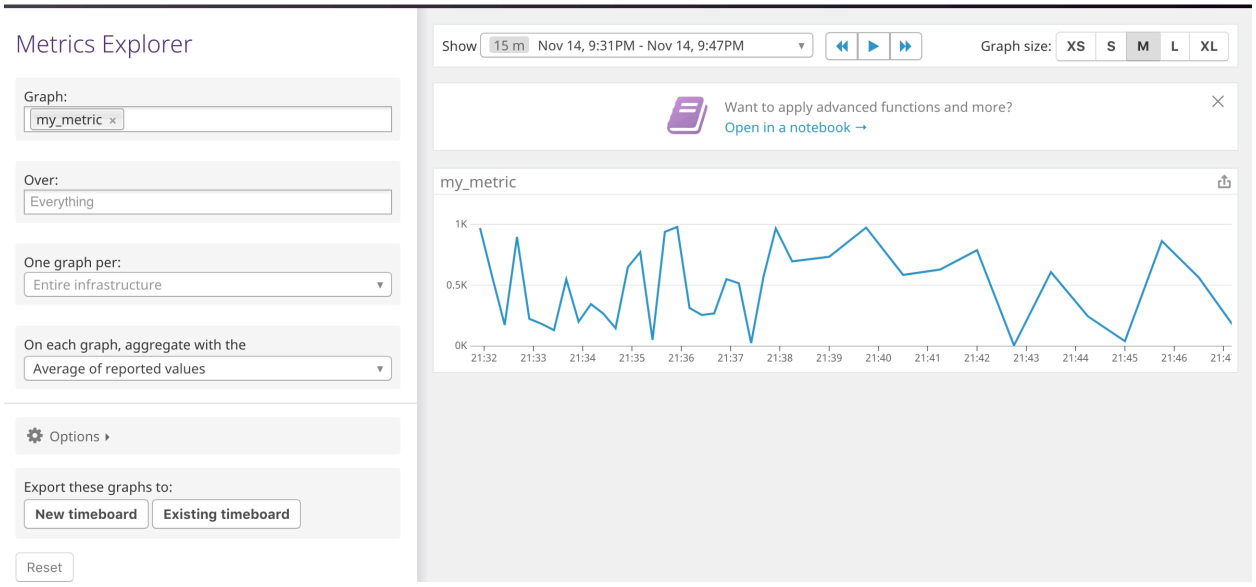
Adding the custom Agent check requires adding two files, the configuration file and the agent check code:

```
/etc/datadog-agent/conf.d/my_check.yaml (configuration file)

init_config:

instances:
    [{}]

/etc/datadog-agent/checks.d/my_check.py. (agent check code)

import random

from checks import AgentCheck
class RandomCheck(AgentCheck):
    def check(self, instance):
        self.gauge('my_metric', random.randint(0,1000))
```

The collector runs every 15-20 seconds. If data isn't needed to be collected that often, the collection interval can be changed. This collection interval can be changed without having to modify the agent check's python code. It is done in the configuration yaml file. Here is the updated yaml that shows the minimum collection interval at 45 seconds:

```
>more /etc/datadog-agent/conf.d/my_check.yaml
init_config:

instances:
  - min_collection_interval: 45
```

Looking at the Metrics Explorer in the UI, we can see the collection interval change. The data points at the left of the graph are at 15 seconds and then start to spread out to 45 seconds on the right.



Using the datadog-agent status command, we can see that our checks are running. (checks highlighted in red)

```
root@ubuntu-xenial:~# datadog-agent status
Getting the status from the agent.
```

```
==============
Agent (v6.6.0)
==============

  Status date: 2018-11-15 05:54:46.693519 UTC
  Pid: 14517
  Python Version: 2.7.15
  Logs:
  Check Runners: 4
  Log Level: info

  Paths
  =====
    Config File: /etc/datadog-agent/datadog.yaml
    conf.d: /etc/datadog-agent/conf.d
    checks.d: /etc/datadog-agent/checks.d

  Clocks
  ======
    NTP offset: 47.657ms
    System UTC time: 2018-11-15 05:54:46.693519 UTC

  Host Info
  =========
    bootTime: 2018-11-14 03:35:49.000000 UTC
    kernelVersion: 4.4.0-139-generic
    os: linux
    platform: ubuntu
    platformFamily: debian
    platformVersion: 16.04
    procs: 179
    uptime: 26h2m29s
    virtualizationRole: guest
    virtualizationSystem: vbox

  Hostnames
  =========
    hostname: ubuntu-xenial
    socket-fqdn: ubuntu-xenial
    socket-hostname: ubuntu-xenial
    hostname provider: os
    unused hostname providers:
      aws: not retrieving hostname from AWS: the host is not an ECS
instance, and other providers already retrieve non-default hostnames
      configuration/environment: hostname is empty
      gce: unable to retrieve hostname from GCE: Get http://
169.254.169.254/computeMetadata/v1/instance/hostname: net/http:
request canceled while waiting for connection (Client.Timeout exceeded
while awaiting headers)
```

```
=========
Collector
=========

  Running Checks
  ==============

    cpu
    ---
        Instance ID: cpu [OK]
        Total Runs: 66
        Metric Samples: 6, Total: 390
        Events: 0, Total: 0
        Service Checks: 0, Total: 0
        Average Execution Time : 1ms


    disk (1.4.0)
    ------------
        Instance ID: disk:e5dffb8bef24336f [OK]
        Total Runs: 65
        Metric Samples: 78, Total: 5,070
        Events: 0, Total: 0
        Service Checks: 0, Total: 0
        Average Execution Time : 29ms


    file_handle
    -----------
        Instance ID: file_handle [OK]
        Total Runs: 66
        Metric Samples: 5, Total: 330
        Events: 0, Total: 0
        Service Checks: 0, Total: 0
        Average Execution Time : 0s


    io
    --
        Instance ID: io [OK]
        Total Runs: 66
        Metric Samples: 52, Total: 3,396
        Events: 0, Total: 0
        Service Checks: 0, Total: 0
        Average Execution Time : 10ms


    load
```

```
----
    Instance ID: load [OK]
    Total Runs: 66
    Metric Samples: 6, Total: 396
    Events: 0, Total: 0
    Service Checks: 0, Total: 0
    Average Execution Time : 0s


memory
------
    Instance ID: memory [OK]
    Total Runs: 66
    Metric Samples: 17, Total: 1,122
    Events: 0, Total: 0
    Service Checks: 0, Total: 0
    Average Execution Time : 0s


my_check (unversioned)
----------------------
    Instance ID: my_check:5ba864f3937b5bad [OK]
    Total Runs: 22
    Metric Samples: 1, Total: 22
    Events: 0, Total: 0
    Service Checks: 0, Total: 0
    Average Execution Time : 0s


network (1.7.0)
---------------
    Instance ID: network:2a218184ebe03606 [OK]
    Total Runs: 66
    Metric Samples: 20, Total: 1,320
    Events: 0, Total: 0
    Service Checks: 0, Total: 0
    Average Execution Time : 0s


ntp
---
    Instance ID: ntp:b4579e02d1981c12 [OK]
    Total Runs: 66
    Metric Samples: 1, Total: 66
    Events: 0, Total: 0
    Service Checks: 1, Total: 66
    Average Execution Time : 41ms
    postgres (2.2.3)
    ----------------
        Instance ID: postgres:b2aa23d9d9f4cf [OK]
        Total Runs: 66
        Metric Samples: 30, Total: 1,980
        Events: 0, Total: 0
        Service Checks: 1, Total: 66
        Average Execution Time : 24ms


    uptime
    ------
        Instance ID: uptime [OK]
        Total Runs: 66
        Metric Samples: 1, Total: 66
        Events: 0, Total: 0
        Service Checks: 0, Total: 0
        Average Execution Time : 0s

========
JMXFetch
========

  Initialized checks
  ==================
    no checks

  Failed checks
  =============
    no checks

=========
Forwarder
=========

  CheckRunsV1: 65
  Dropped: 0
  DroppedOnInput: 0
  Events: 0
  HostMetadata: 0
  IntakeV1: 6
  Metadata: 0
  Requeued: 0
  Retried: 0
  RetryQueueSize: 0
  Series: 0
  ServiceChecks: 0
  SketchSeries: 0
  Success: 136
  TimeseriesV1: 65

  API Keys status
  ===============
    API key ending with 050b8 on endpoint https://app.datadoghq.com:
API Key valid

==========
Logs Agent
==========

  postgres
  --------
    Type: file
    Path: /var/lib/postgresql/9.5/main/pg_log/pg.log
    Status: OK
    Inputs: /var/lib/postgresql/9.5/main/pg_log/pg.log

=========
DogStatsD
=========

  Checks Metric Sample: 15,391
  Event: 1
  Events Flushed: 1
  Number Of Flushes: 65
  Series Flushed: 13,137
  Service Check: 749
  Service Checks Flushed: 803
  Dogstatsd Metric Sample: 418
```

Visualizing data is done through graphing. Dashboards with the UI are made up of these graphs. Automation is key to being able to manage large environments and data. The APIs available with Datadog allow you to easily script or integrate with other tools. Utilizing the Datadog API, a Timeboard will be created that contains three graphs:

- The custom metric scoped over your host.
- The Postgres Rows Fetched metric with the anomaly function applied.
- The custom metric with the rollup function applied to sum up all the points for the past hour into one bucket

The API script will be implemented using python. Scripts can be written in other languages. The API is well documented with examples in Python, Ruby, and Curl. For Python, a datadog package can be installed and used:

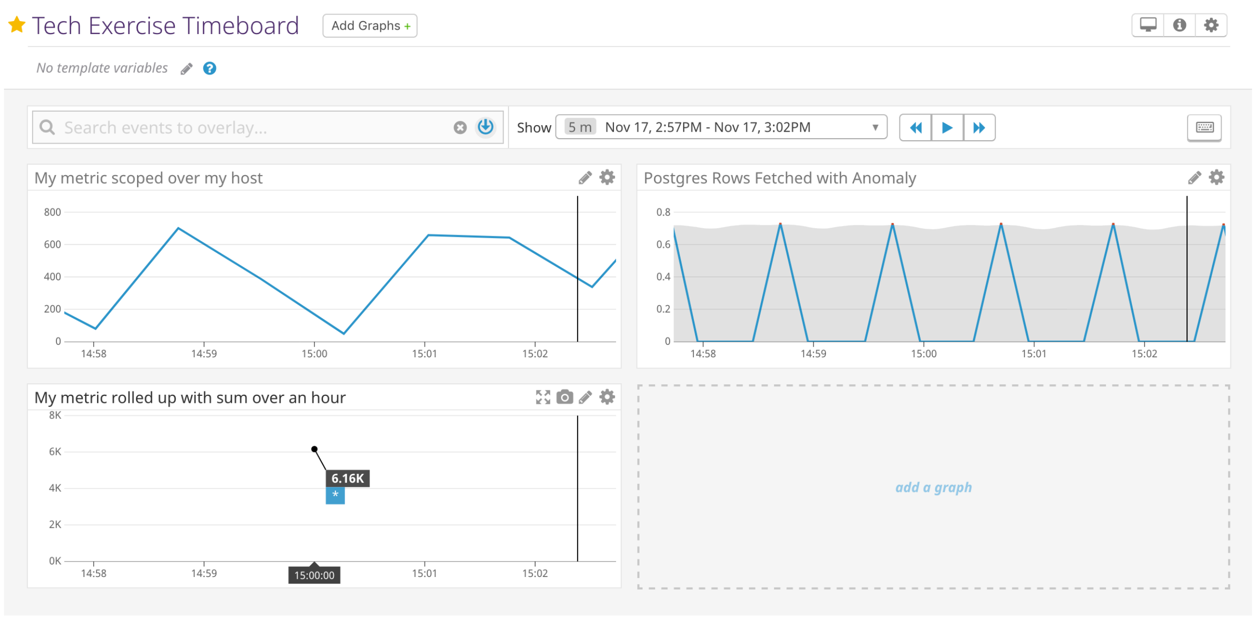
```
pip install datadog
```

The script used for this exercise is:

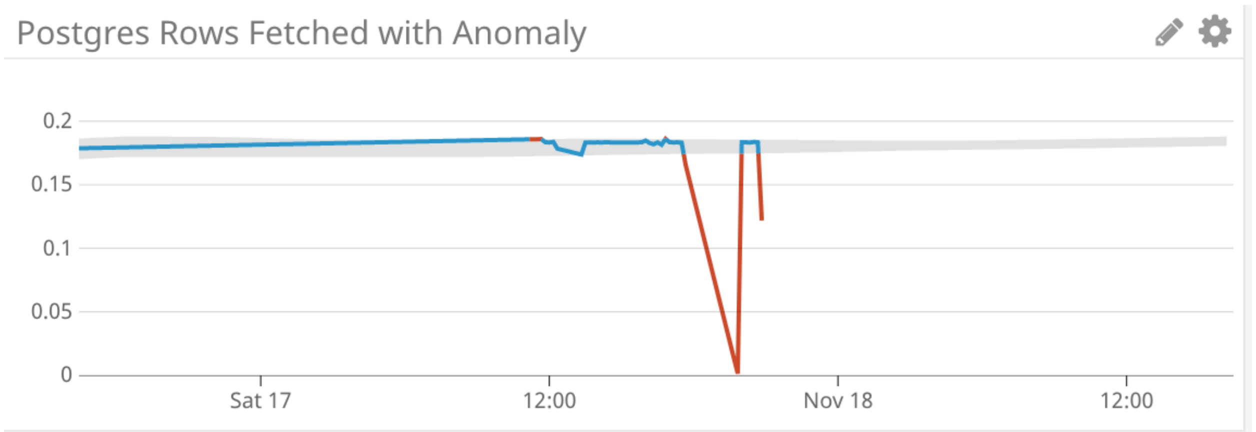
```
#!/opt/datadog-agent/embedded/bin/python

from datadog import initialize, api

options = {
    'api_key': '48ff5cf94558d3471524bf7d894050b8',
    'app_key': 'e5b1c6040602a7cc34dbdba3033ee8cf52cff9df'
}

initialize(**options)

title = "Tech Exercise Timeboard"
description = "The timeboard for the tech exercise."
graphs = [{
    "definition": {
        "events": [],
         "requests": [
           {"q": "avg:my_metric{*}"}
         ],
        "viz": "timeseries"
    },
    "title": "My metric scoped over my host"
},
{
    "definition": {
        "events": [],
        "requests": [
            {"q": "anomalies(avg:postgresql.rows_fetched{*}, 'basic', 2)"}
        ],
        "viz": "timeseries"
    },
    "title": "Postgres Rows Fetched with Anomaly"
},
{
    "definition": {
        "events": [],
        "requests": [
            {"q": "avg:my_metric{*}.rollup(sum,3600)"}
        ],
        "viz": "timeseries"
    },
    "title": "My metric rolled up with sum over an hour"
 }]
```

```
read_only = True
api.Timeboard.create(title=title,
                     description=description,
                     graphs=graphs,
                     read_only=read_only)
```

After running this script the following dashboard is created. The timeframe shown on the graphs has been scoped to 5 minutes.


Tech Exercise Timeboard
Add Graphs +
No template variables
Search events to overlay...
Show
5 m
Nov 17, 2:57PM - Nov 17, 3:02PM
My metric scoped over my host
Postgres Rows Fetched with Anomaly
My metric rolled up with sum over an hour
6.16K
15:00:00
add a graph

The anomaly graph is designed to identify when a metric is behaving differently than it has behaved in the past. The algorithm takes into account trends, day of week, and time of day patterns. Anomaly graphs use past data to make the predictions so the new graph above does not indicate an anomaly. The shading shows normal behavior. Allowing for longer time, the below graph shows anomalous behavior indicated by the red line:



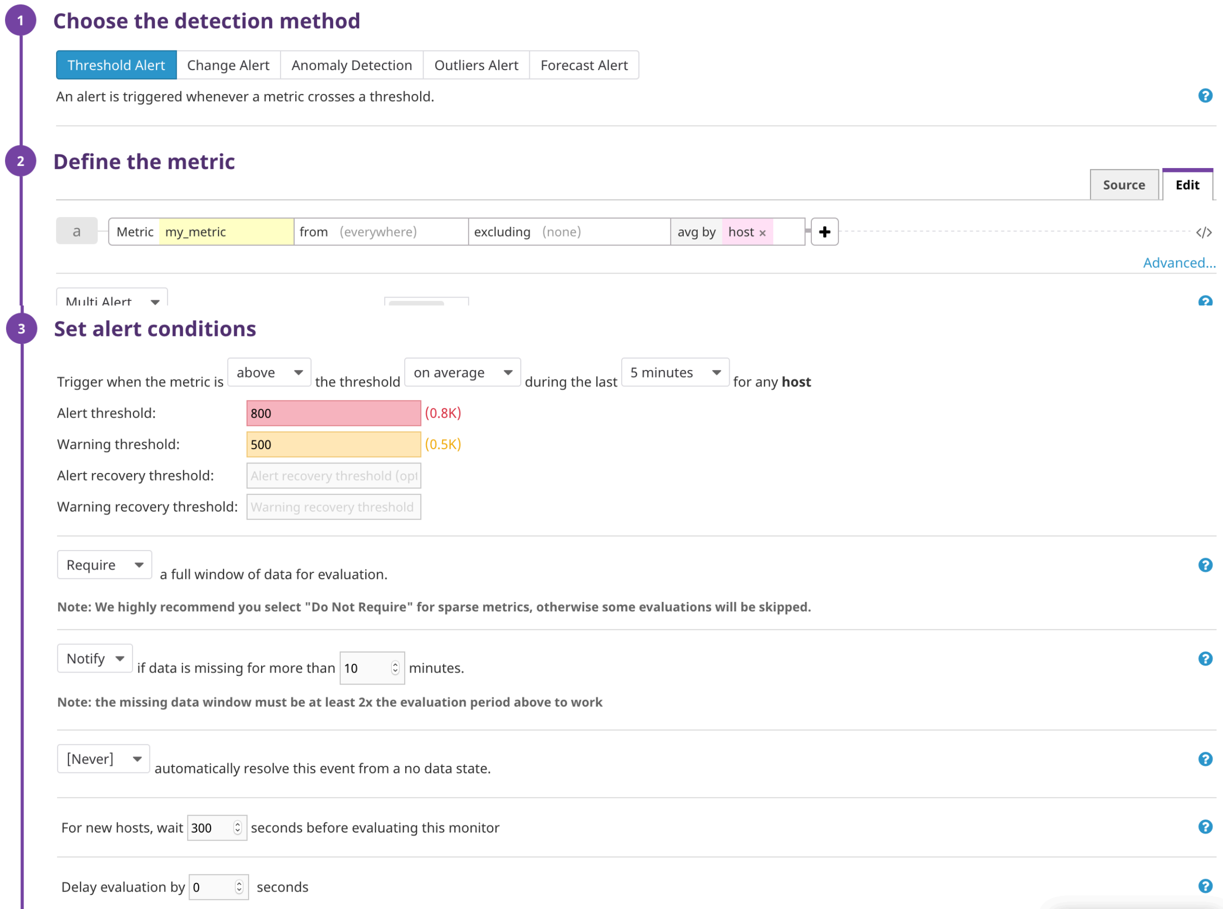
Getting alerted when issues occur is critical in being able to address them quickly. Monitors can be used do this. They can be configured for warnings, alerts, and no data. It can send out an email to you, team members, 3rd party services, or custom endpoints via webhooks. The following monitor was configured on the custom metric ‘my_metric’. It will alert if it’s above the following values over the past 5 minutes:

- Warning threshold of 500
- Alerting threshold of 800
- if there is No Data for this query over the past 10m

The monitor’s message will:

- Send 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.

The monitor configuration from the UI is:



1 Choose the detection method

An alert is triggered whenever a metric crosses a threshold.

2 Define the metric

a Metric my_metric from (everywhere) excluding (none) avg by host ×

3 Set alert conditions

Trigger when the metric is above the threshold on average during the last 5 minutes for any **host**

Alert threshold: 800 (0.8K)

Warning threshold: 500 (0.5K)

Alert recovery threshold: Alert recovery threshold (opt

Warning recovery threshold: Warning recovery threshold

Require a full window of data for evaluation.

Note: We highly recommend you select "Do Not Require" for sparse metrics, otherwise some evaluations will be skipped.

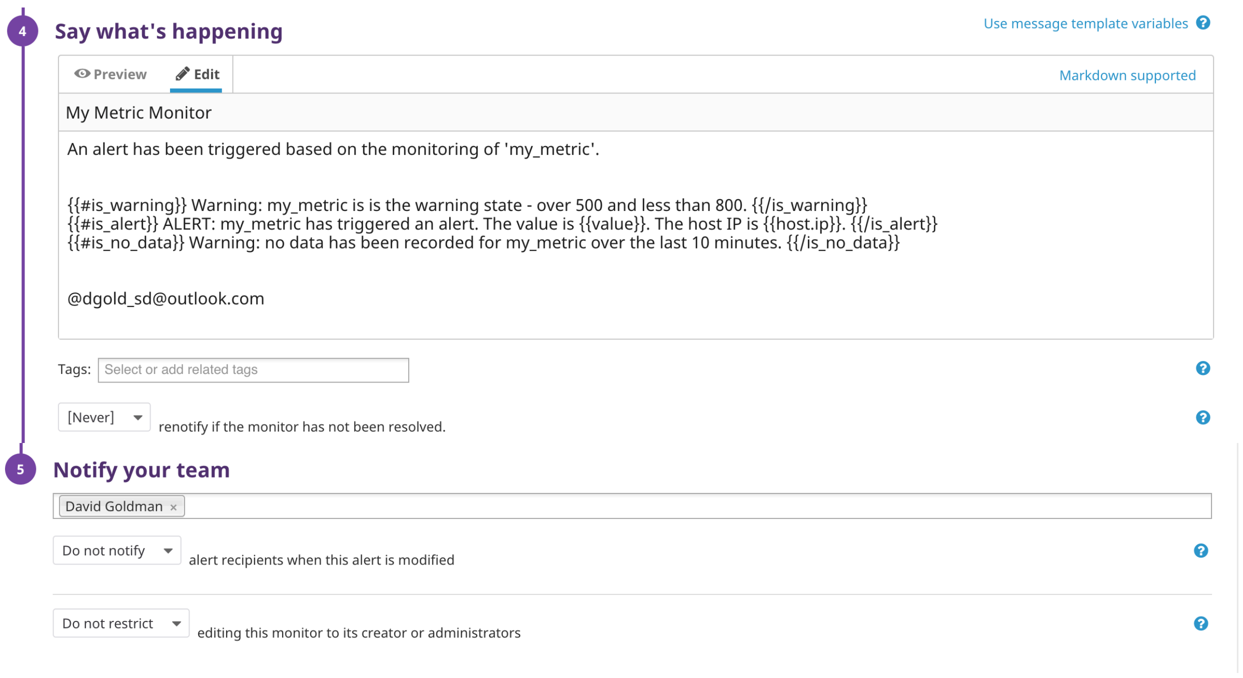
Notify if data is missing for more than 10 minutes.

Note: the missing data window must be at least 2x the evaluation period above to work

[Never] automatically resolve this event from a no data state.

For new hosts, wait 300 seconds before evaluating this monitor

Delay evaluation by 0 seconds



4 Say what's happening

Use message template variables

Preview | Edit

Markdown supported

My Metric Monitor

An alert has been triggered based on the monitoring of 'my_metric'.

{{#is_warning}} Warning: my_metric is is the warning state - over 500 and less than 800. {{/is_warning}}
{{#is_alert}} ALERT: my_metric has triggered an alert. The value is {{value}}. The host IP is {{host.ip}}. {{/is_alert}}
{{#is_no_data}} Warning: no data has been recorded for my_metric over the last 10 minutes. {{/is_no_data}}

@dgold_sd@outlook.com

Tags: Select or add related tags

[Never] renotify if the monitor has not been resolved.

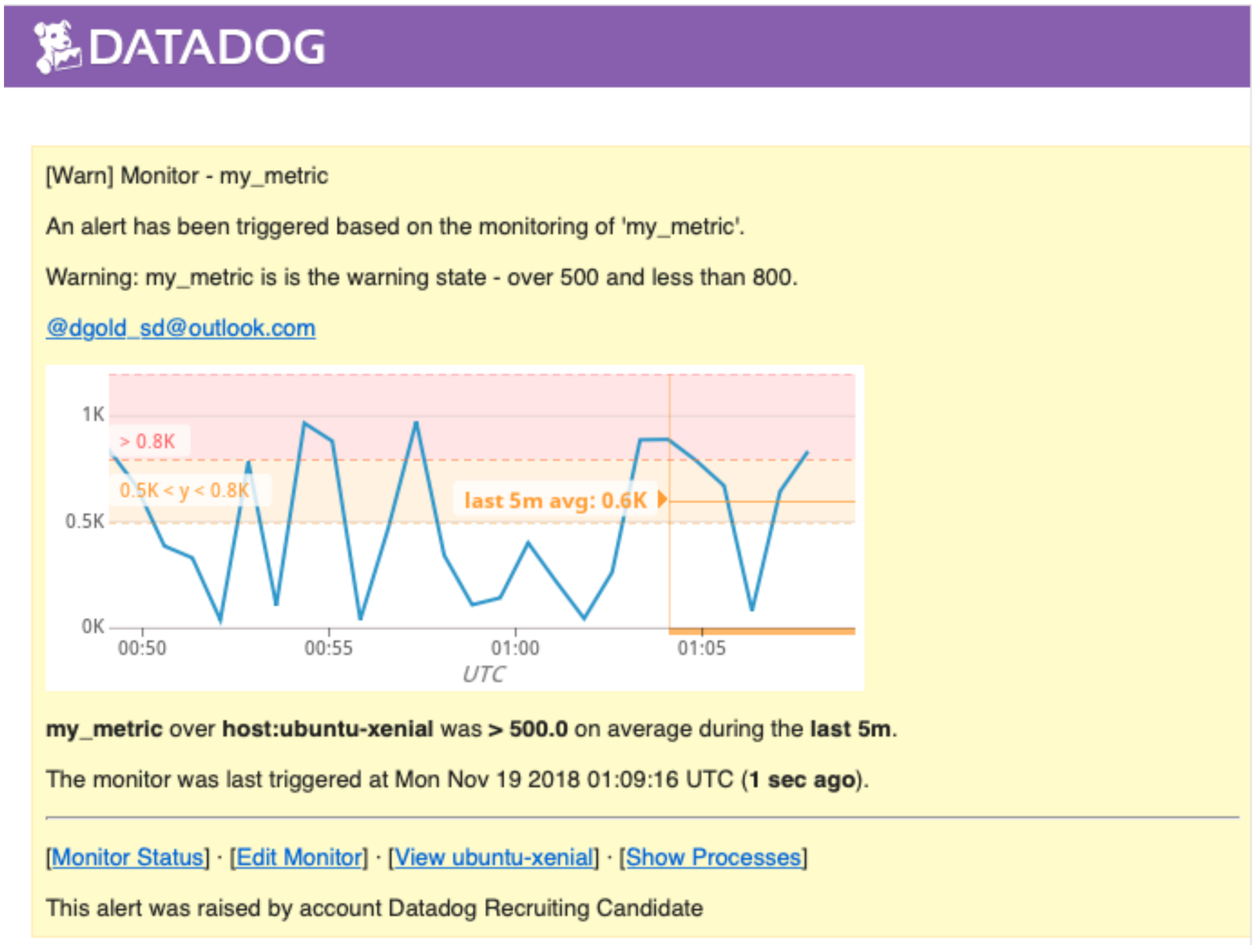
5 Notify your team

David Goldman ×

Do not notify alert recipients when this alert is modified

Do not restrict editing this monitor to its creator or administrators

The email alert, a warning, that was sent is:



DATADOG

[Warn] Monitor - my_metric

An alert has been triggered based on the monitoring of 'my_metric'.

Warning: my_metric is is the warning state - over 500 and less than 800.

@dgold_sd@outlook.com

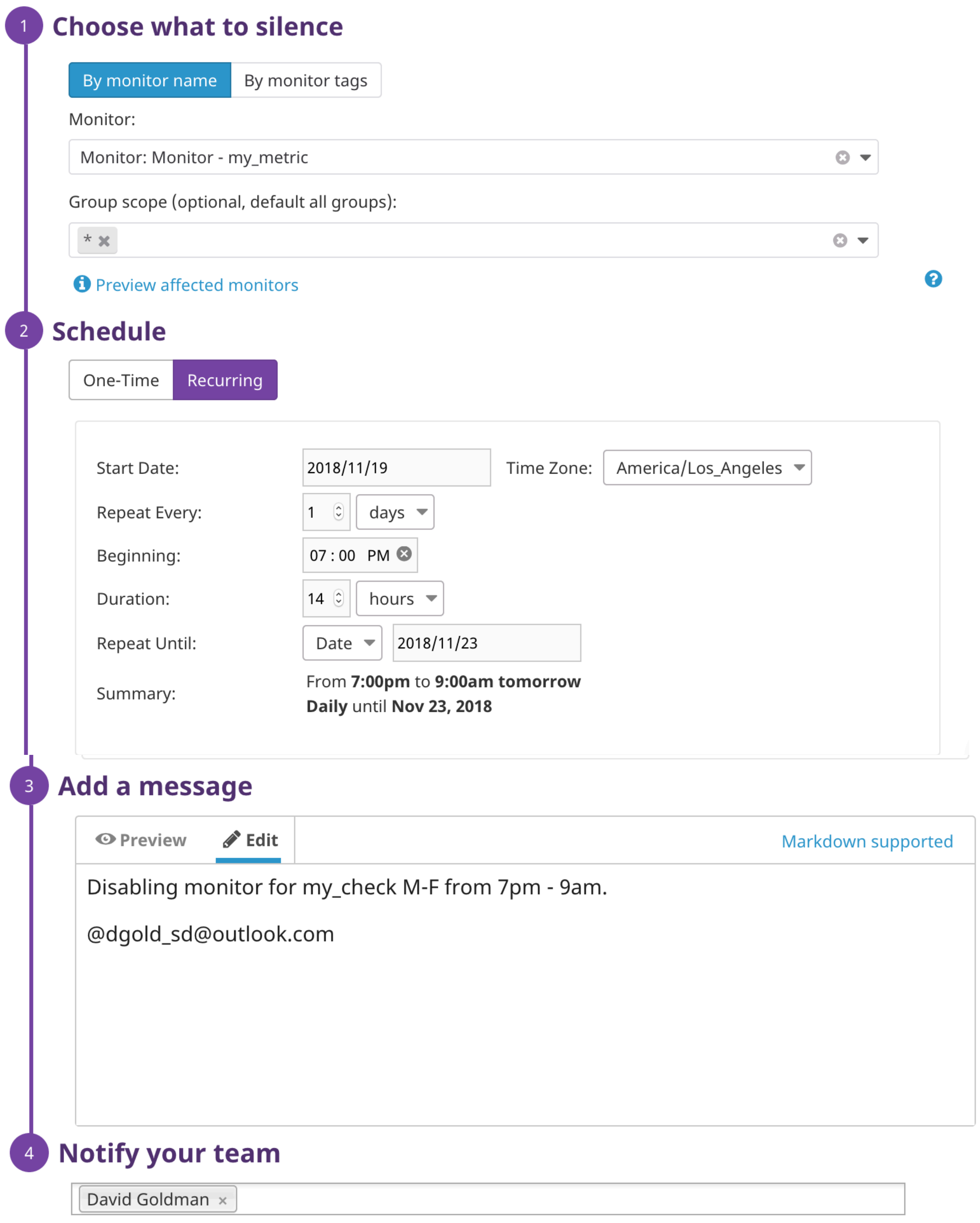
my_metric over **host:ubuntu-xenial** was **> 500.0** on average during the **last 5m**.

The monitor was last triggered at Mon Nov 19 2018 01:09:16 UTC (**1 sec ago**).

[Monitor Status] · [Edit Monitor] · [View ubuntu-xenial] · [Show Processes]

This alert was raised by account Datadog Recruiting Candidate

Monitors can be muted manually so that alerts are disabled until unmuted. It is also possible to schedule downtime. This may be useful if systems are scheduled to be taken offline and alerts would be unnecessary. The following shows the configuration that silences alerts from 7pm to 9am daily, M-F, and one that silences the monitoring for the rest of the day on Sunday, from the time of this exercise:



1 Choose what to silence

[✓ By monitor name] By monitor tags

Monitor: Monitor: Monitor - my_metric

Group scope (optional, default all groups): *

Preview affected monitors

2 Schedule

One-Time [✓ Recurring]

Start Date: 2018/11/19 Time Zone: America/Los_Angeles

Repeat Every: 1 days

Beginning: 07 : 00 PM

Duration: 14 hours

Repeat Until: Date 2018/11/23

Summary: From **7:00pm** to **9:00am tomorrow**
Daily until **Nov 23, 2018**

3 Add a message

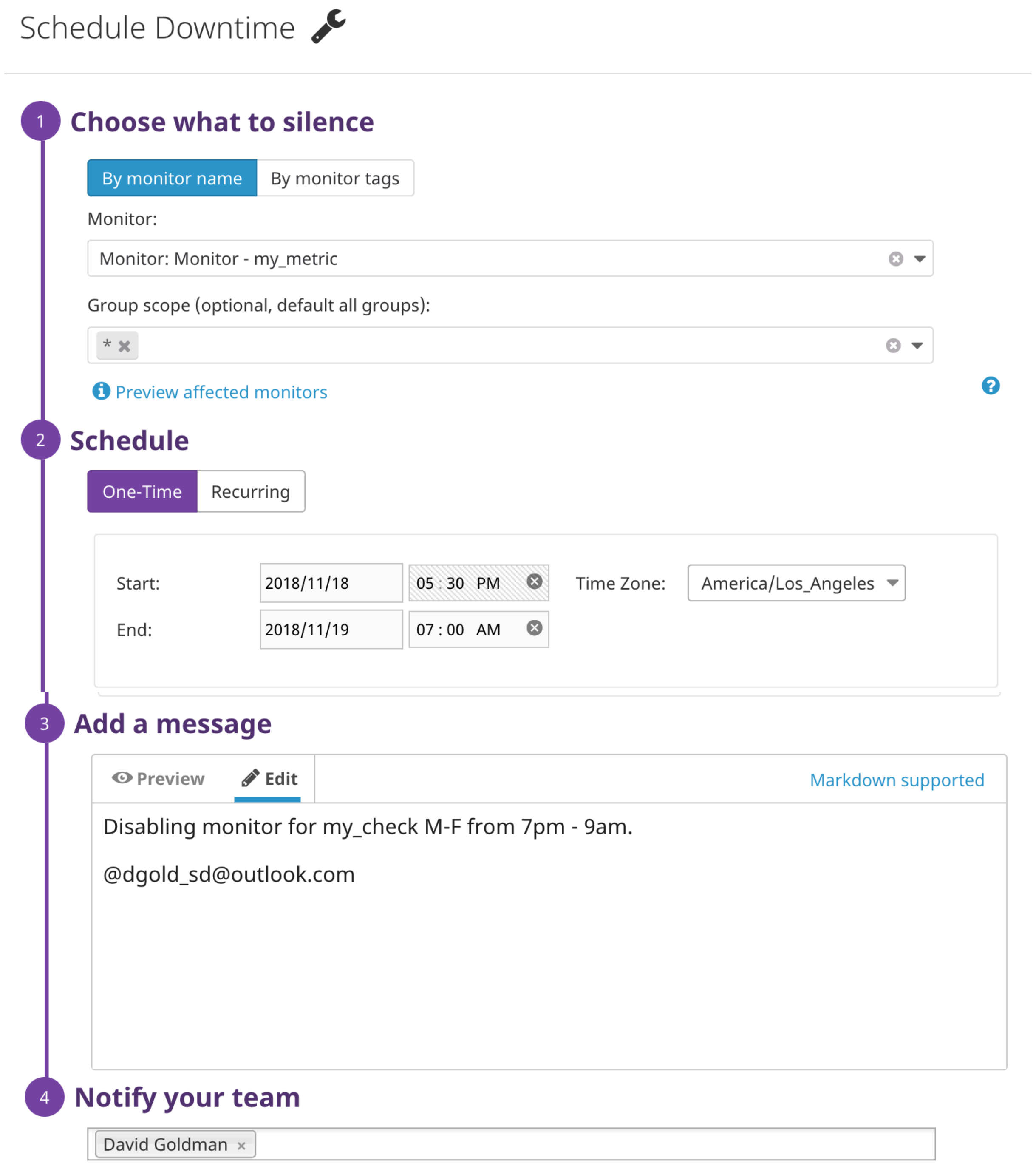
Preview Edit Markdown supported

Disabling monitor for my_check M-F from 7pm - 9am.

@dgold_sd@outlook.com

4 Notify your team

David Goldman



Schedule Downtime

1 Choose what to silence

By monitor name | By monitor tags

Monitor:

Monitor: Monitor - my_metric

Group scope (optional, default all groups):

*

Preview affected monitors

2 Schedule

One-Time | Recurring

Start:	2018/11/18	05 : 30 PM	Time Zone:	America/Los_Angeles
End:	2018/11/19	07 : 00 AM		

3 Add a message

Preview | Edit | Markdown supported

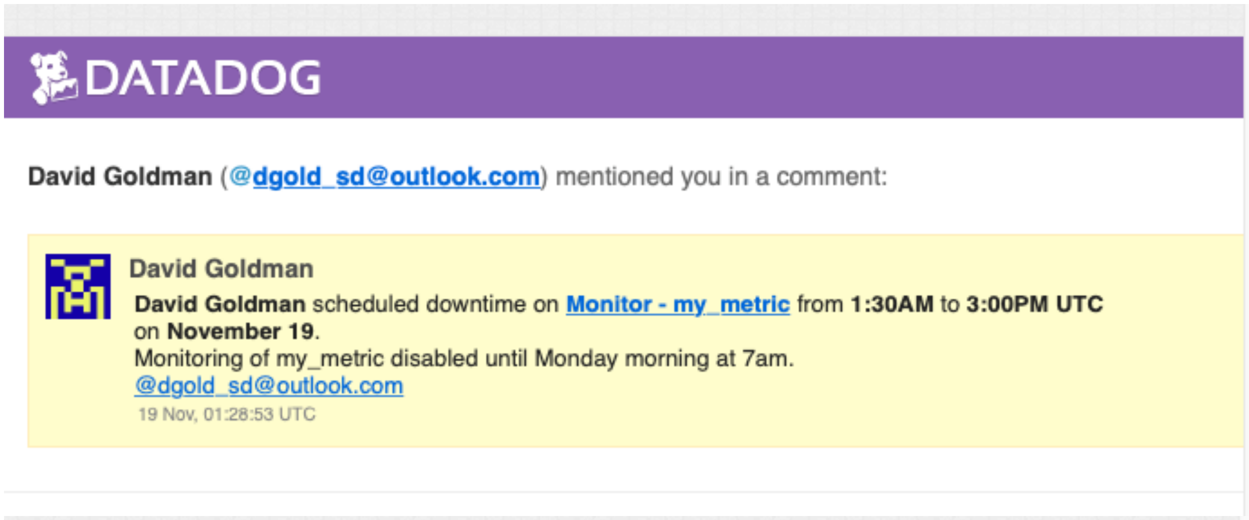
Disabling monitor for my_check M-F from 7pm - 9am.

@dgold_sd@outlook.com

4 Notify your team

David Goldman ×

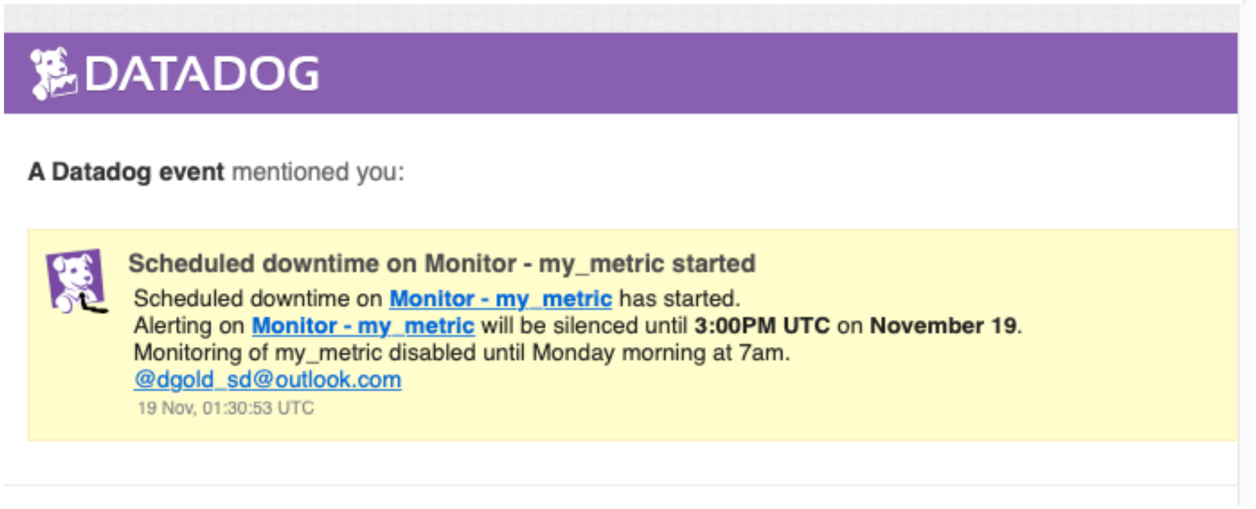
The email received when scheduling and starting downtime is:



DATADOG

David Goldman (@dgold_sd@outlook.com) mentioned you in a comment:

David Goldman
David Goldman scheduled downtime on **Monitor - my_metric** from **1:30AM** to **3:00PM UTC** on **November 19**.
Monitoring of my_metric disabled until Monday morning at 7am.
@dgold_sd@outlook.com
19 Nov, 01:28:53 UTC



DATADOG

A Datadog event mentioned you:

Scheduled downtime on Monitor - my_metric started
Scheduled downtime on **Monitor - my_metric** has started.
Alerting on **Monitor - my_metric** will be silenced until **3:00PM UTC** on **November 19**.
Monitoring of my_metric disabled until Monday morning at 7am.
@dgold_sd@outlook.com
19 Nov, 01:30:53 UTC

Application Performance Monitoring (APM) provides insights into an application's performance. It is easy to get started with APM using the ddtrace-run command. The following python example flask application was used in this test:

```
> more apm_test.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(__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='10.0.2.15', port='5050')
```

This application requires two packages to be installed:

```
/opt/datadog-agent/embedded/bin/pip install flask
/opt/datadog-agent/embedded/bin/pip install blinker
```

The application was then run using ddtrace-run:

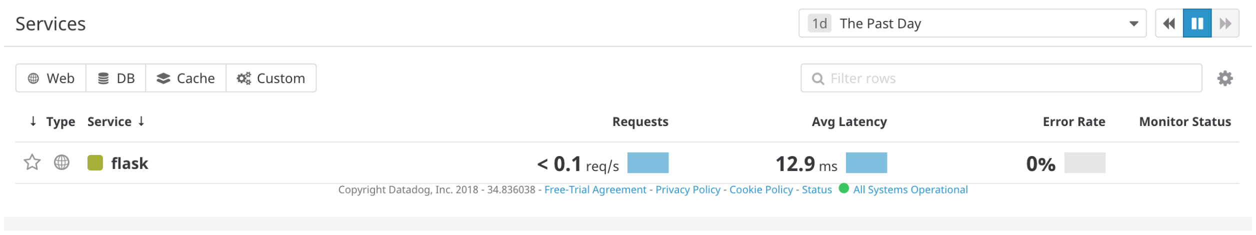
```
/opt/datadog-agent/embedded/bin/ddtrace-run /opt/datadog-agent/
embedded/bin/python apm_test.py &
```

To generate some data, curl was used to hit the entry points:

```
curl http://10.0.2.15:5050
curl http://10.0.2.15:5050/api/apm
curl http://10.0.2.15:5050/api/trace
```

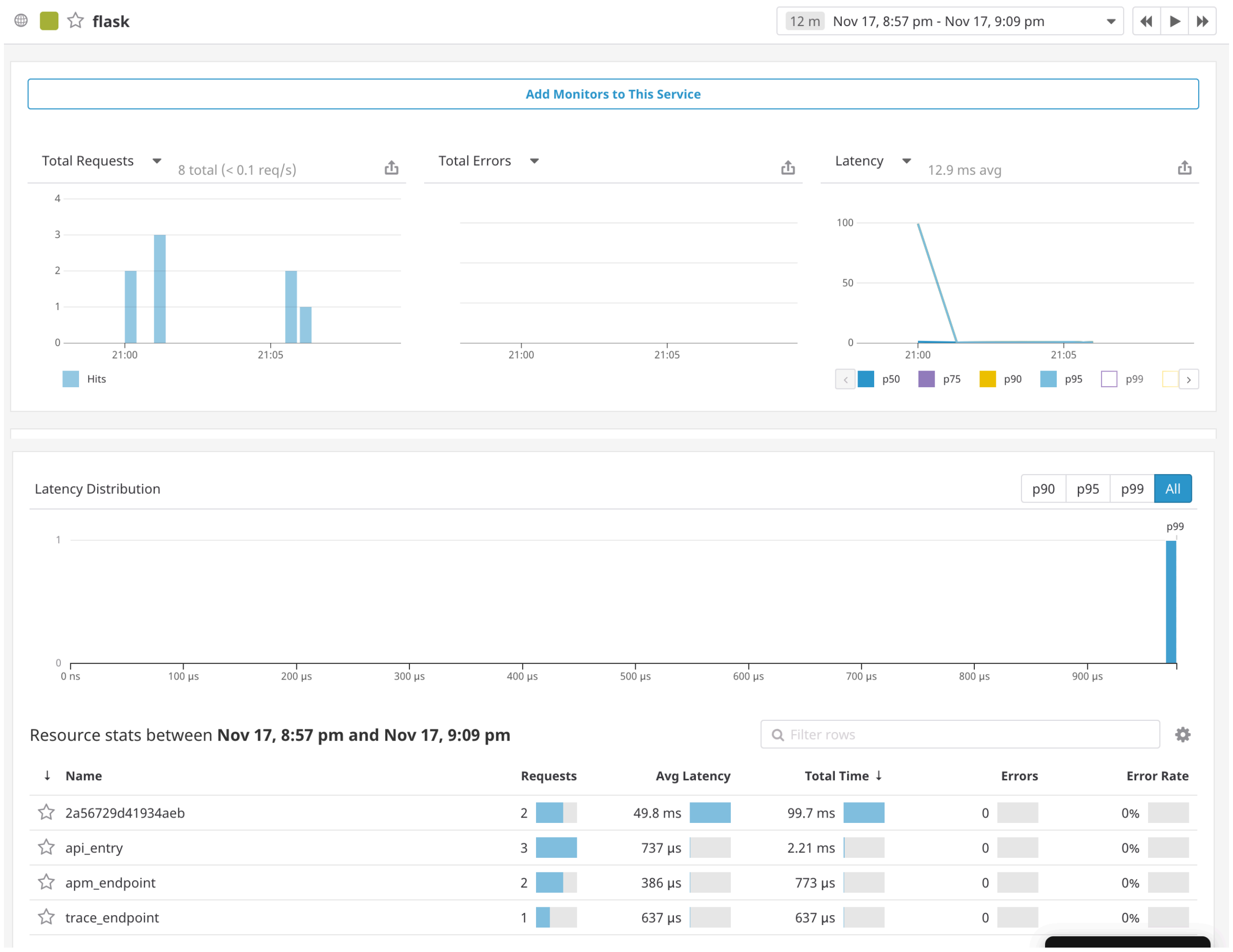
The results can be seen in the UI at

https://app.datadoghq.com/apm/services?end=1542592224080&env=dave&paused=false&start=1542505824080&watchdogOnly=false


Services

1d The Past Day

Type	Service	Requests	Avg Latency	Error Rate	Monitor Status
🌐	🟩 flask	< 0.1 req/s	12.9 ms	0%	



Resource stats between **Nov 17, 8:57 pm and Nov 17, 9:09 pm**

Name	Requests	Avg Latency	Total Time	Errors	Error Rate
2a56729d41934aeb	2	49.8 ms	99.7 ms	0	0%
api_entry	3	737 µs	2.21 ms	0	0%
apm_endpoint	2	386 µs	773 µs	0	0%
trace_endpoint	1	637 µs	637 µs	0	0%

From the screenshots you can see two different terms used - services and resources. Services are a set of processes that do the same job. A resource is an action for a service. In this example, the main flask app is a service. It is a webapp service. Other services may be databases, caches, app servers, etc. Applications can consist of multiple services, for example a simple 3-tier application may include web server, app server, and database services. Resources will vary by service. In this example, the URLs for the application - api_entry, apm_endpoint, and trace_endpoint - are all examples of resources of the service.

Finally, there are a lot of people out there who like to fish. Spending the day out on the water is nice but coming home with dinner is even better. There have been too many days when I've come home empty handed. There are a lot of factors that go into finding the fish but I'm sure some captains are better at finding the schools than others. The fish reports I've seen here in San Diego show the boats, number of anglers, and catch by species. This is usually posted daily. It would be nice to monitor this and look for trends over time to find that ideal time, boat, and captain. A dashboard that showed trends with good and bad days, morning vs afternoon, trends across boats, and types of fish caught would be a useful place for most fisherman. Downloading the data, parsing it, and reporting it to Datadog with a custom Agent check would be ideal. Of course, it might be hard to track the big ones that got away.