

SOLUTIONS ENGINEER TECHNICAL EXERCISE

Chris Reites

October 2018



BEFORE WE GET STARTED...



Here's a little bit about me. My name is Chris Reites, and I'm currently in presales at CA Technologies helping to sell their DevOps/Continuous Testing software solutions. Prior to joining CA, I spent 16 years at FedEx as a software developer, QA Engineer, and Enterprise Architect. Having spent a lot of time as a customer of enterprise software, I believe I have great perspective and appreciation for the challenges that my customers face. The reason why I made the move into presales, and why I enjoy it so much, is that I love being able to combine the excitement of sales with my passion for helping customers solve their problems and bring more value into their own organizations.

Now I'm looking for my next adventure. That's where you come in! I've been talking to people I know at Datadog, and it sounds like an amazing place to be. Even before I knew about this technical exercise, I signed up for a free trial at [datadoghq.com](https://www.datadoghq.com) and started playing around with the technology. I was highly impressed and that's what ultimately convinced me that I wanted to join your team. My hope is that after reviewing my submission, you'll feel the same about me.

When I'm not busy helping customers with their digital transformations, I enjoy living in the Tampa area with my wife and two children. On the weekends you can usually find us boating or at the beach. I also like running, but to ensure I don't become *too* healthy, I also enjoy brewing my own beer.

<https://www.linkedin.com/in/chris-reites/>

MY ENVIRONMENT

For this exercise, I chose to use an AWS instance I already had setup. It's running Ubuntu 16.04 and I have a few java apps running there that I was able to play around with to check out more Datadog functionality. I previously had configured my applications to use SQL Server as the backend database, but because the current version of that integration doesn't work on Linux, I installed MySQL and changed my apps to use that for the backend. Finally, in order to create some interesting moments that I could look at metrics for, I used BlazeMeter, which is a cloud-based load generation solution, to create a lot of traffic to my applications.





Section 1

COLLECTING METRICS

The first thing I needed to do was get the Datadog Agent stood up on my server, and configure it with a meaningful name and some helpful tags. To do this, I simply had to edit the datadog.yaml file found in /etc/datadog-agent

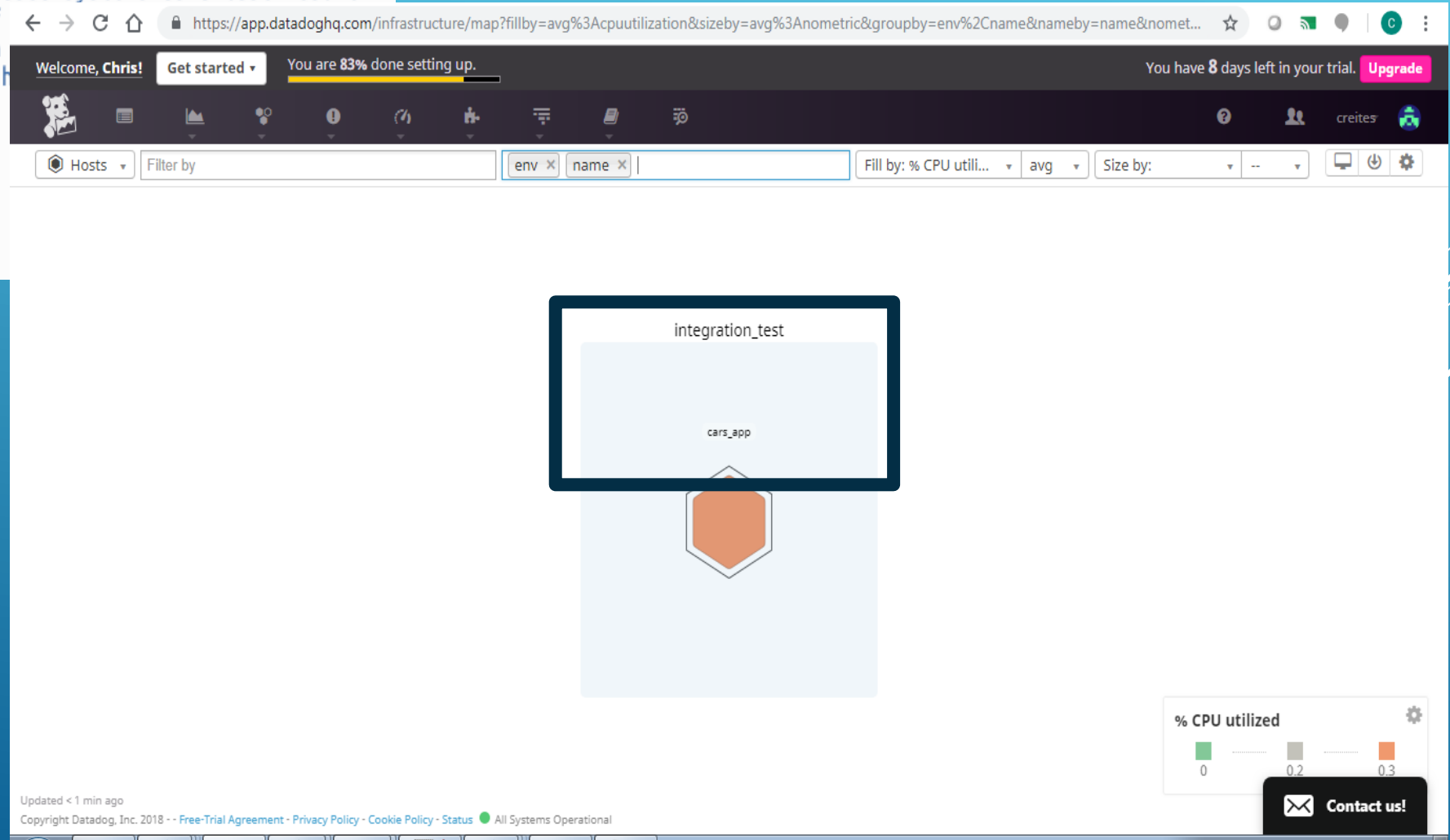


ubuntu@ip-172-31-8-44: /etc/datadog-agent

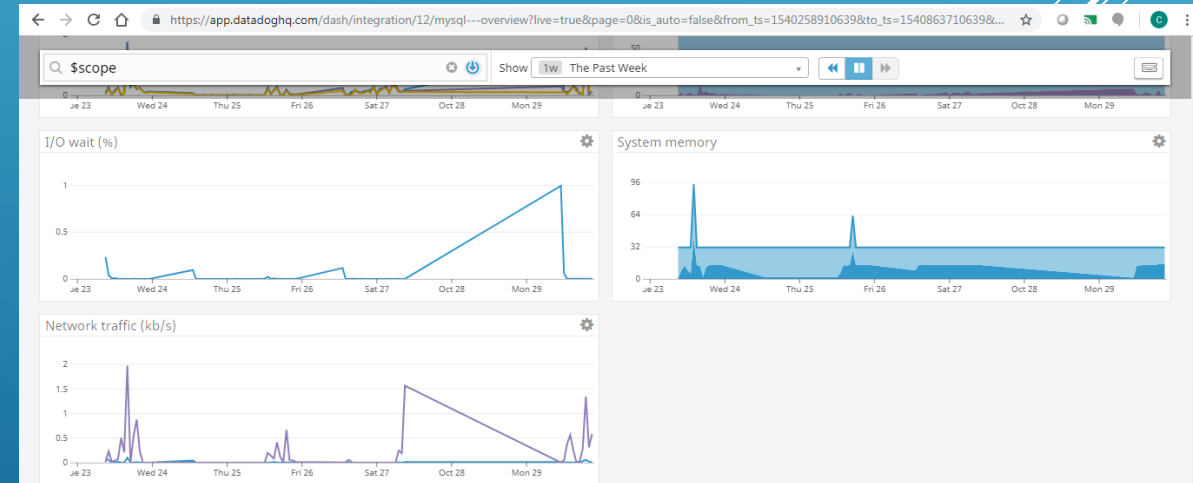
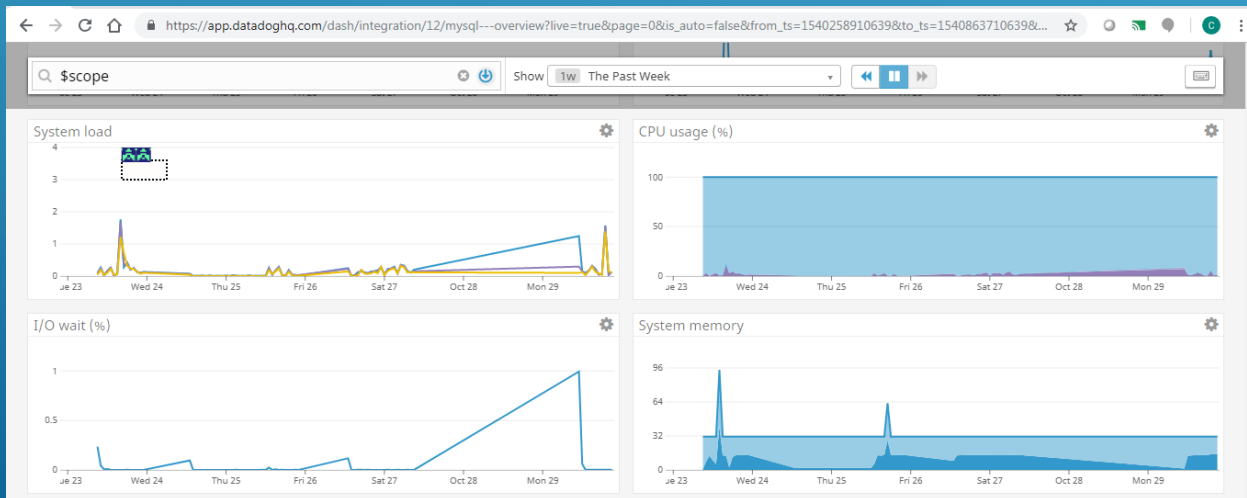
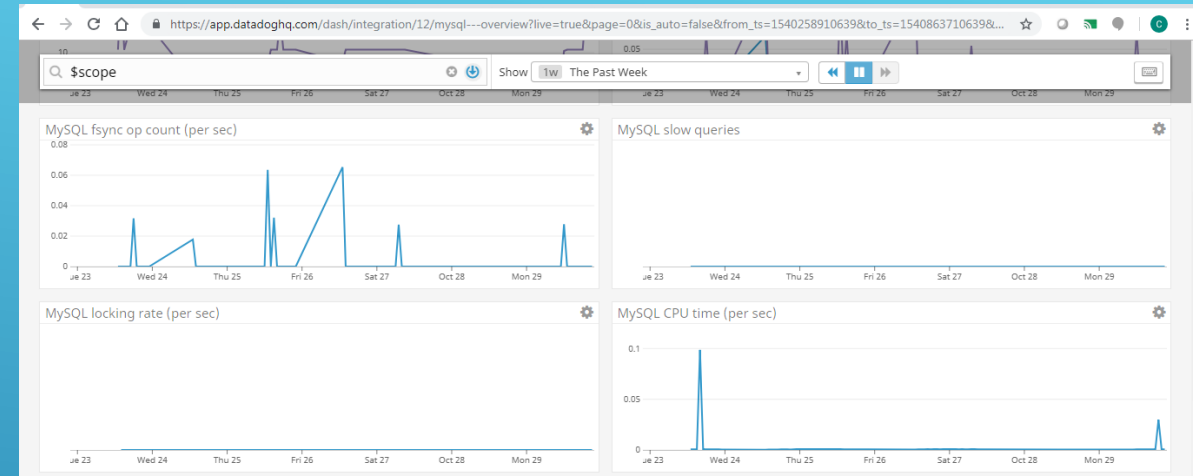
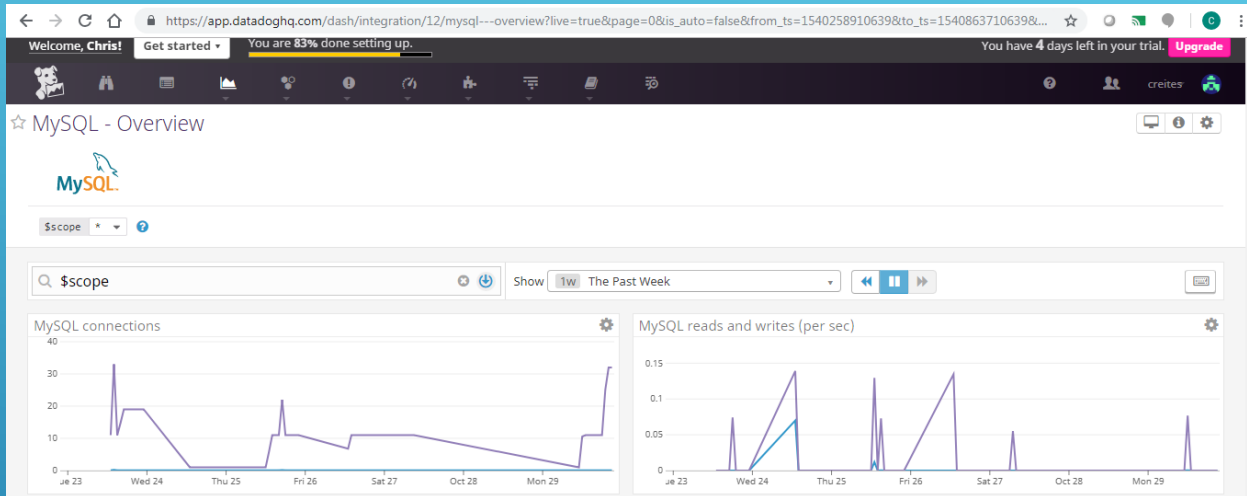
File Edit View Search Terminal Help

```
# Make the agent use "hostname -f" on unix-based systems as a last resort
# way of determining the hostname instead of
# This will be enabled by default in version
# More information at https://dtdg.co/flag-h
# hostname_fqdn: false
```

```
# Set the host's tags (optional)
tags:
  - name:cars_app
  - env:integration_test
```



Next I added in the MySQL integration so that I could keep an eye on my database metrics. One of the nice features of Datadog is that when you install integrations, you get default Dashboards which include popular metrics. Here are some shots of my default MySQL Dashboard. It's pretty easy to spot when I was running my load tests!



Speaking of being easy to spot... When you do see something that catches your eye, it's very easy to collaborate with your team. Here, I created a snapshot of one of the spikes, annotated it, and then sent it off for the team to look into. Unfortunately, I'm the only one on my team, so it was up to me to figure out what was going on and reply. Sometimes it feels like I have to do everything!



←

→

↻

🏠

https://app.datadoghq.com/event/event?id=4634529501735448578

☆

🔌

💬

🟢

⋮

Welcome, Chris!

Get started ▾

You are 83% done setting up.

You have 8 days left in your trial. Upgrade

creites

Datadog Event

System load

System load

4

3

2

1

0

Mon 22

Tue 23

Wed 24

Thu 25

UTC-0400

Please take a look at what caused this spike on the system around 4pm Tuesday. Thanks.

Thu Oct 25 2018 17:19:10 GMT-0400 (Eastern Daylight Time) · View · Lower priority · Edit

creites@gmail.com

Nevermind, that was when we were running a load test on the system, so it was to be expected.

just now · Edit

Leave a comment...

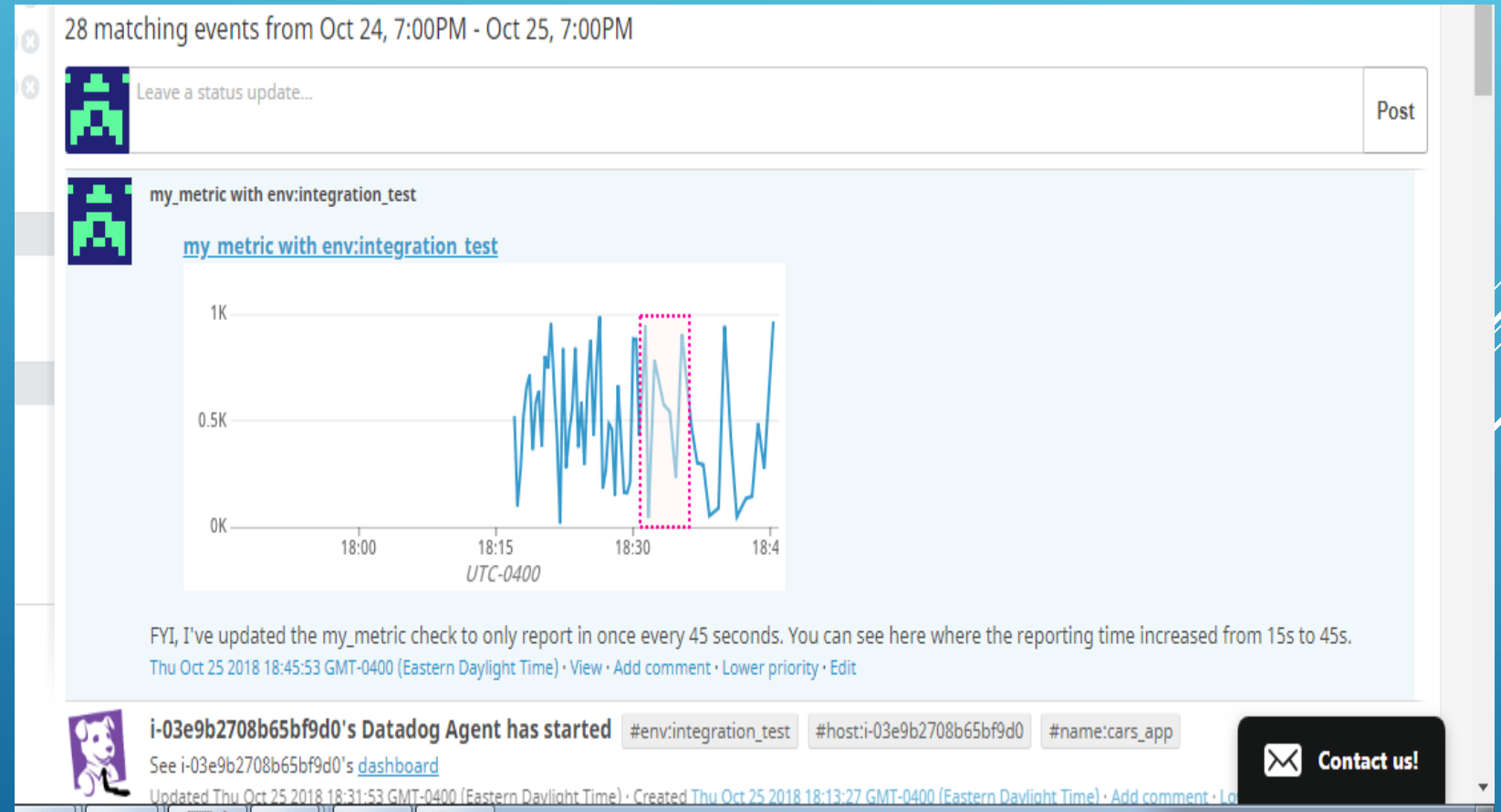
Post

In order to further test out the metric gathering capabilities of Datadog, I wrote a simple python script that generates a random number between 1 and 1000 and then submits the value to the Agent with the name `my_metric`. After setting up the script, I then changed the collection interval to be every 45 seconds instead of the default 15 seconds.



```
ubuntu@ip-172-31-8-44: /  
File Edit View Search Terminal Help  
__version__="1.0.0"  
import time  
import random  
  
from checks import AgentCheck  
  
class DemoCheck(AgentCheck):  
    def check(self, instance):  
        random_nbr = random.randint(1,1001)  
        self.gauge('my_metric', random_nbr)  
~  
~
```

```
ubuntu@ip-172-31-8-44: /  
File Edit View Search Terminal Help  
init_config:  
  
instances:  
  - min_collection_interval: 45  
~  
~  
~
```



For fun (and bonus points!) I added in the ability to dynamically change the collection interval by simply updating a delay file on the filesystem. Whenever the check is fired, it will re-read the delay file and skip reporting the metric if the collection interval hasn't passed.



```
ec2-18-191-165-227.us-east-2.compute.amazonaws.com - Remote Desktop Connection
ubuntu@ip-172-31-8-44:/etc/datadog-agent/checks.d$ cat CheckDemo.py
__version__="1.0.0"
import time
import random

from checks import AgentCheck

class CheckDemo(AgentCheck):
    def check(self, instance):
        #Open our delay file and read in the current collection interval
        delay_file = open('/etc/datadog-agent/checks.d/CheckDemo.delay', 'r')
        collect_interval = int(delay_file.read())

        #The first time we run the check, we need to create the last_collection attribute
        #By creating it as current time - the collection interval, we guarantee we run
        #the check the first time
        if not hasattr(self, 'last_collection'):
            self.last_collection = time.time() - collect_interval

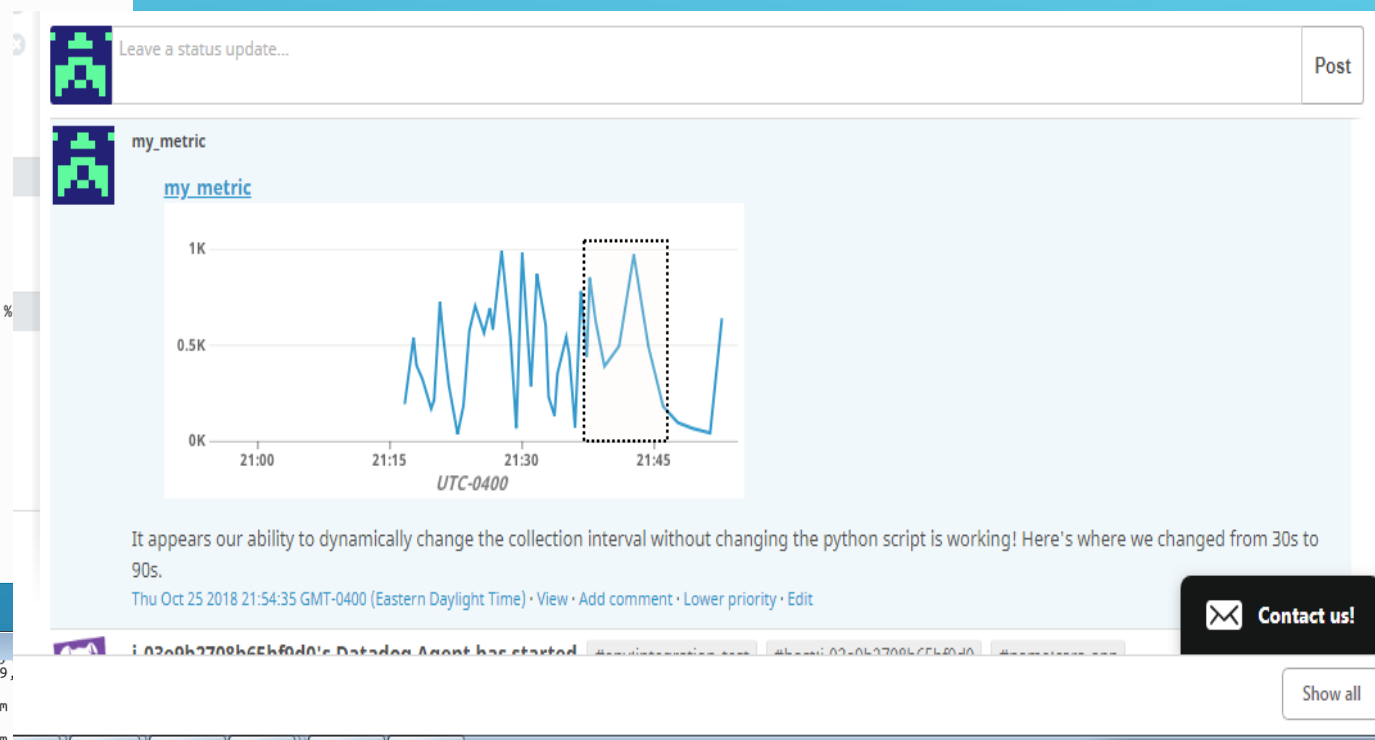
        #Check to see if we've already submitted the check within the collection interval
        if time.time() - self.last_collection <= collect_interval:
            #If we're not going to submit the check, send a log message
            self.log.info('Not submitting check because custom collection interval not met. Current collection interval is set to: %s' % collect_interval)

            delay_file.close()
            return

        delay_file.close()
        #Since we're going to submit the metric, update the last collection time
        self.last_collection=time.time()

        #Get our random number and call the gauge method to send in our metric
        random_nbr = random.randint(1,1001)
        self.gauge('my_metric', random_nbr)

ubuntu@ip-172-31-8-44:/etc/datadog-agent/checks.d$ cat CheckDemo.delay
90
ubuntu@ip-172-31-8-44:/etc/datadog-agent/checks.d$
```



```
ec2-18-191-165-227.us-east-2.compute.amazonaws.com - Remote Desktop Connection
t\n record.message = record.getMessage()\n File \"/opt/datadog-agent/embedded/lib/python2.7/logging/__init__.py", line 329,\nsg = msg % self.args\nTypeError: not all arguments converted during string formatting\n]]
2018-10-26 01:38:07 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom\not met. Current collection interval is set to: 30
2018-10-26 01:38:22 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom\not met. Current collection interval is set to: 30
2018-10-26 01:38:52 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 30
2018-10-26 01:39:07 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 30
2018-10-26 01:39:37 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 90
2018-10-26 01:39:52 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 90
2018-10-26 01:40:07 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 90
2018-10-26 01:40:22 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection\not met. Current collection interval is set to: 90
2018-10-26 01:40:37 UTC | INFO | (datadog_agent.go:151 in LogMessage) | (CheckDemo.py:18) | Not submitting check because custom collection
```



Section 2

VISUALIZING DATA

The Datadog platform includes very powerful APIs that can be used to programmatically create many of the same assets you can build manually inside the web frontend. One example is Timeboards. Here is a python script I wrote to create a custom Timeboard.



```
ubuntu@ip-172-31-8-44: ~/python_scripts
File Edit View Search Terminal Help
ubuntu@ip-172-31-8-44:~/python_scripts$ cat create_timeboard.py
from datadog import initialize, api

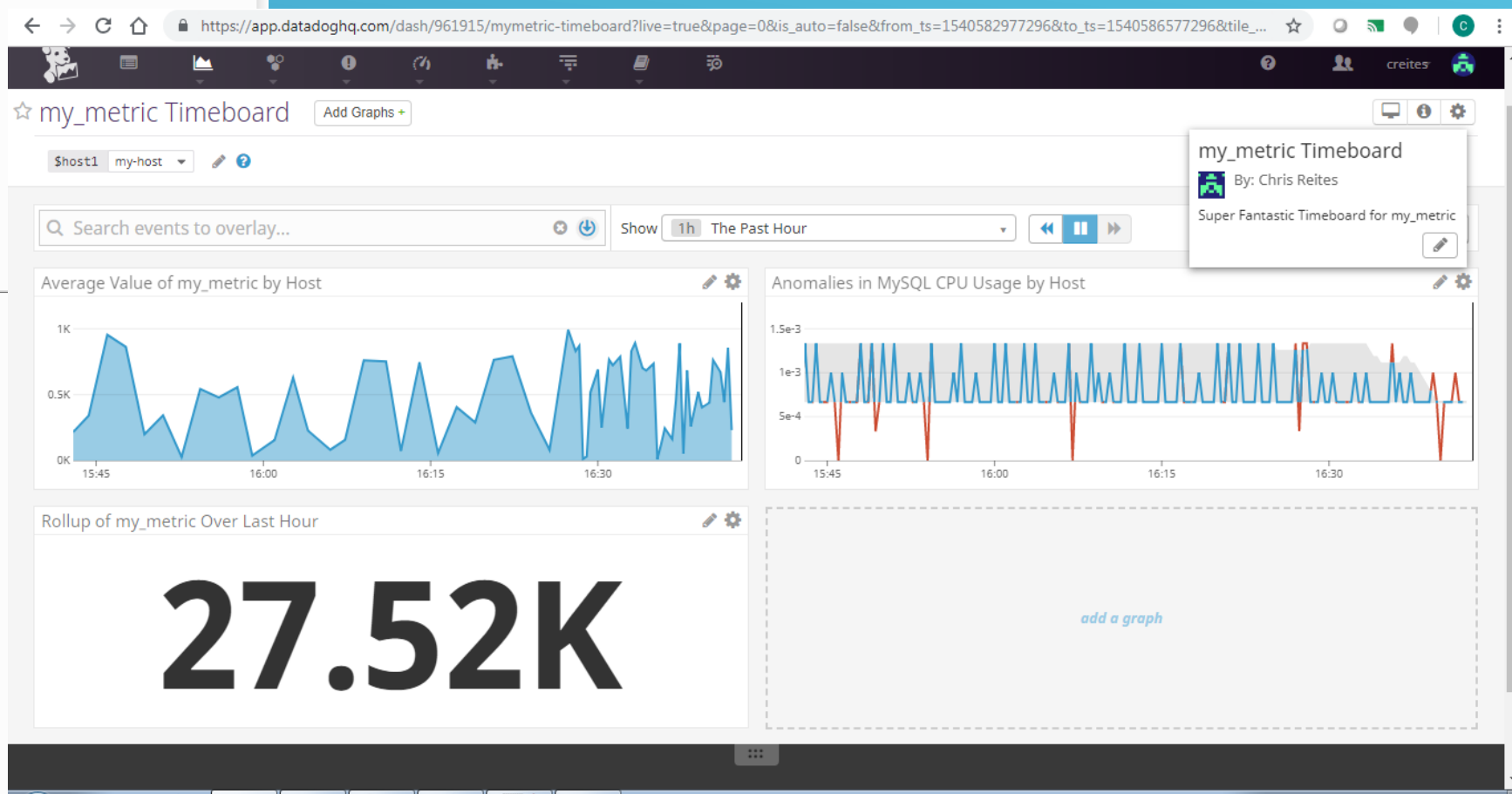
# Set our api and app keys
# Note that if you accidentally type api-key instead of api_key
# you will lose 90 minutes and half your mind trying to figure
# out why the initialize call keeps failing. Ask me how I know.
options = {
    'api_key': 'a4b4f014a7018',
    'app_key': 'be34903eeacd6'
}

initialize(**options)

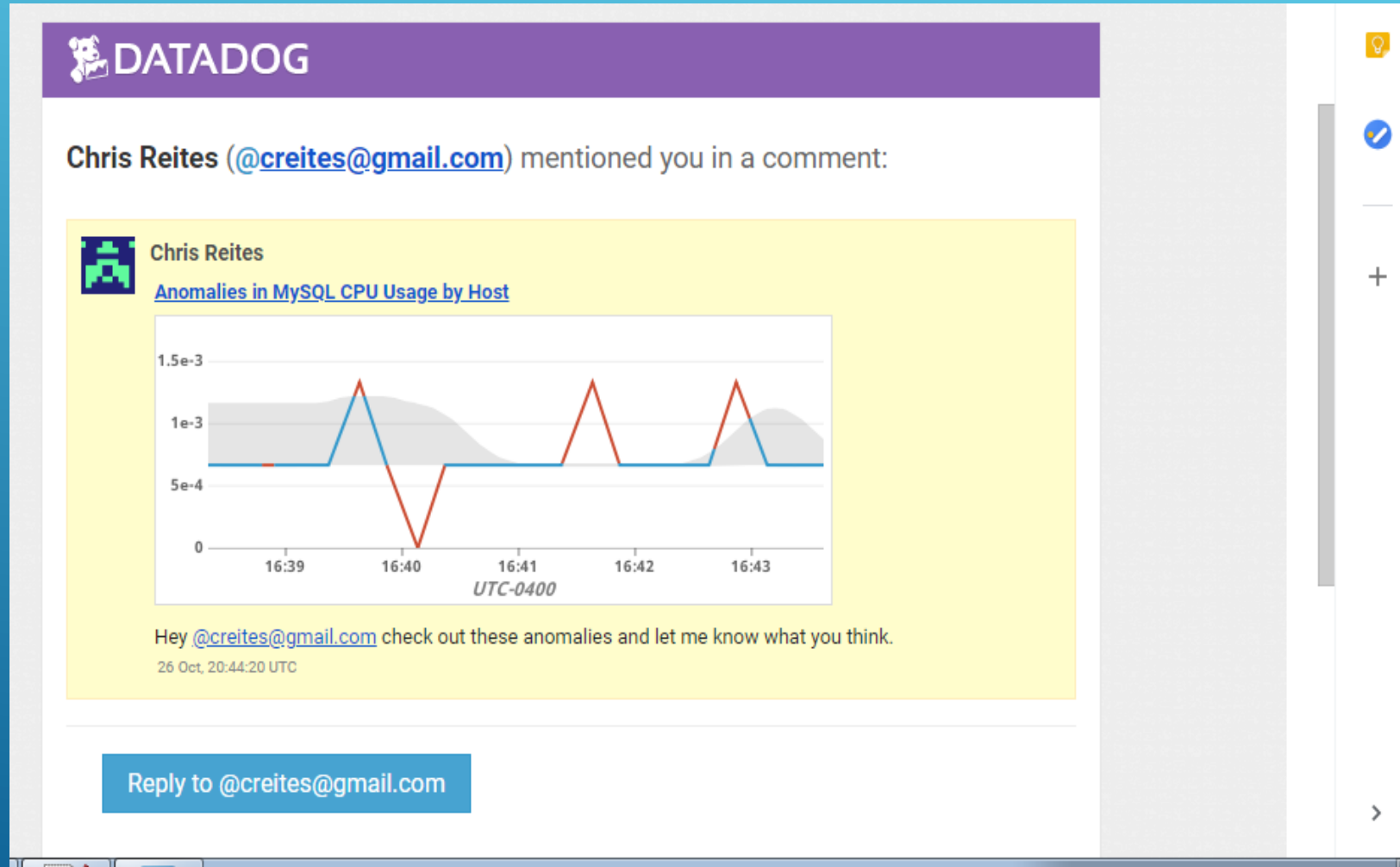
title = 'my_metric Timeboard'
description = 'Super Fantastic Timeboard for my_metric'
graphs = [
    # Our first graph will be the average value of my_metric
    {
        'definition': {
            'events': [],
            'requests': [
                {'q': 'avg:my_metric{*} by {hosts}'}
            ],
            'viz': 'timeseries'
        },
        'title': 'Average Value of my_metric by Host'
    },
    # Our second graph will show the cpu usage of our MySQL database
    {
        'definition': {
            'events': [],
            'requests': [
                {'q': "anomalies(avg:mysql.performance.cpu_time{*}, 'basic', 2)"}
            ],
            'viz': 'timeseries'
        },
        'title': 'Anomalies in MySQL CPU Usage by Host'
    },
    # Finally, let's add a query_value to our timeboard that shows the sum
    # of the my_metric values over the last hour.
    {
        'definition': {
            'events': [],
            'requests': [
                {'q': 'avg:my_metric{*}.rollup(sum,3600)'}
            ],
            'viz': 'query_value',
            'status': 'done'
        },
        'title': 'Rollup of my_metric Over Last Hour'
    }
]

# Set some basic variables for our timeboard
template_variables = [
    {
        'name': 'host1',
        'prefix': 'host',
        'default': 'host:my-host'
    }
]

read_only=True
# Call the api to create the timeboard
api.Timeboard.create(
    title=title,
    description=description,
    graphs=graphs,
    template_variables=template_variables,
    read_only=read_only
)
ubuntu@ip-172-31-8-44:~/python_scripts$
```



Here's an email containing a snapshot that I generated from my custom Timeboard. In this case, I'm looking at a 5 minute window of my graph that shows Anomalies in the CPU usage of MySQL. The Anomalies function is powerful because it compares actual metrics to expected values derived from historical trend analysis.





Section 3

MONITORING DATA

One of the most useful aspects of the Datadog platform is the ability to create monitors for anything you want to be notified about. In this example, I created a monitor to alert me with either a warning or an alert if my custom random number metric rose above an average of either 500 or 800 over a 5 minute period. I'll also be notified if my metric doesn't send in any data over the last 10 minutes.



← → ↺ ⌂ https://app.datadoghq.com/monitors#6856098/edit ☆ 🔊 🔍

1 Choose the detection method

Threshold Alert Change Alert Anomaly Detection Outliers Alert Forecast Alert

An alert is triggered whenever a metric crosses a threshold. ?

2 Define the metric

Source Edit

a Metric **my_metric** from host:i-03e9b2708b65bf9d0 x excluding (none) avg by (everything) + </> Advanced...

Simple Alert ▼ Trigger a single alert when your metric satisfies your alert conditions. ?

3 Set alert conditions

Trigger when the metric is above ▼ the threshold on average ▼ during the last 5 minutes ▼

Alert threshold: 800 (0.8K)

Warning threshold: 500 (0.5K)

Alert recovery threshold: Alert recovery threshold (optional)

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.

Contact us!

When setting up monitors, you can create highly customized messages that vary depending on things such as warnings vs alerts. You can also include variables such as the host IP address and metric values to make the monitor emails more meaningful.



← → ↺ ↻ ⌂ https://app.datadoghq.com/monitors#6856098/edit

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

Delay evaluation by 0 seconds

4 Say what's happening

Preview Edit

Issue with metric: my_metric on host: {{host.ip}}

```
{{#is_warning}}
Please be advised that the average value of my_metric over the last 5 minutes has crossed above WARN level on host {{host.ip}}. Current average value of
my_metric over the last 5 minutes is: {{value}}
{{/is_warning}}
{{#is_alert}}
Please be advised that the average value of my_metric over the last 5 minutes has crossed above ALERT level on host {{host.ip}}. Current average value c
my_metric over the last 5 minutes is: {{value}}
{{/is_alert}}
{{#is_no_data}}
Please be advised that my_metric has not reported data for the last 10 minutes on host {{host.ip}}
{{/is_no_data}} @creites@gmail.com
```

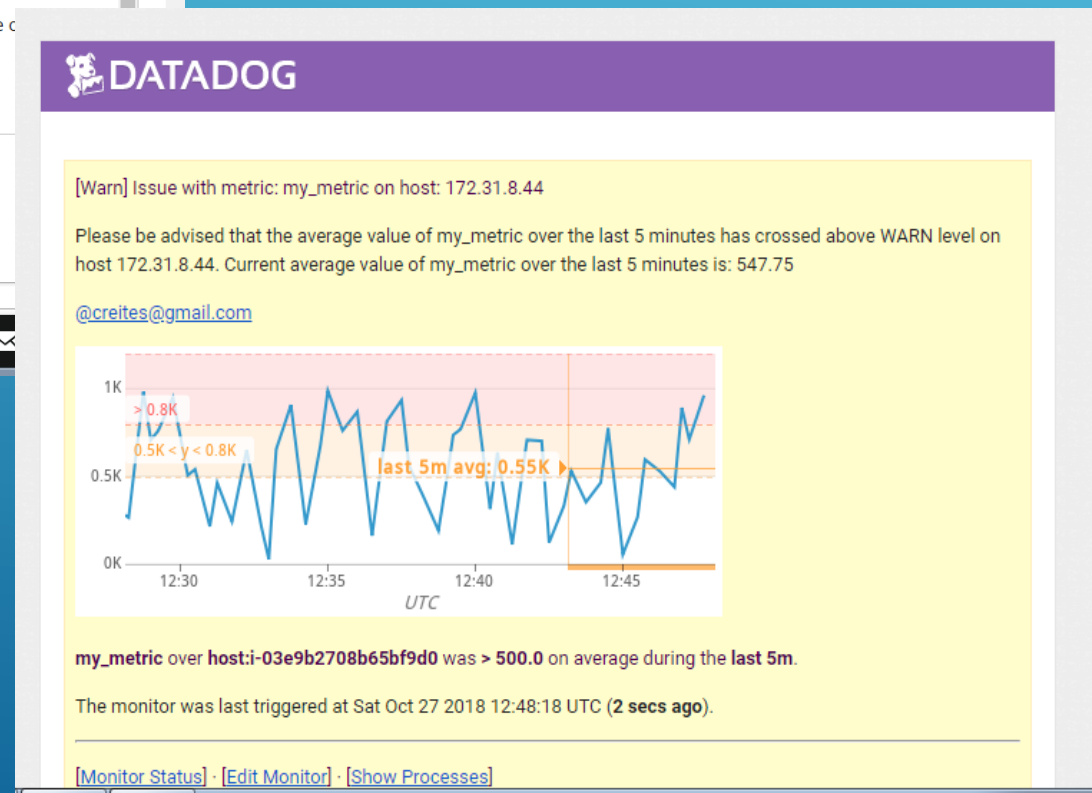
Tags: Select or add related tags

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

5 Notify your team

Chris Reites x

Do not notify alert recipients when this alert is modified



And because I don't want to be bothered while watching football on Saturdays or Sundays, it's always great to be able to schedule downtime windows when the monitors will automatically mute themselves. Here are two that when combined will mute the monitor on weekday evenings and also Saturday starting at 9am through Monday at 9am. We can also make sure the system sends us an email when it mutes a monitor.



https://app.datadoghq.com/monitors#downtime?id=410310924

Welcome, Chris! Get started You are 83% done setting up.

Triggered Monitors Manage Monitors Manage Downtime

Q Filter downtime

STATUS 1	SCOPE	MONITOR TAGS	MON
SCHEDULED	+	+	Issue {{host.ip}}
RECURRING	+	+	Issue {{host.ip}}
SCHEDULED	+	+	Issue {{host.ip}}
RECURRING	+	+	Issue {{host.ip}}
ACTIVE	+	+	One

Downtime

Scope: *

Monitor: Issue with metric: my_metric on host: {{host.ip}}

Scheduled to start Oct 29, 2018 19:00 EDT and repeats daily from 7:00pm to 9:00am tomorrow

Scheduled by Chris Reites

Please be advised that the monitor for my_metric will be silenced from 7pm until 9am. @creites@gmail.com

Showing 1 result

STATUS	NAME	DEFINITION	TAGS
WARN	Issue with metric: my_metric on host: {{host.ip}}	my_metric	

DATADOG

A Datadog event mentioned you:

Scheduled downtime on Issue with metric: my_metric on host: {{host.ip}} started

Scheduled downtime on [Issue with metric: my_metric on host: {{host.ip}}](#) has started. Alerting on [Issue with metric: my_metric on host: {{host.ip}}](#) will be silenced until 1:00PM UTC on October 29. Please be advised that the monitor for my_metric will be silenced from 9am Saturday until 9am Monday. @creites@gmail.com

27 Oct, 13:00:31 UTC

Reply on Datadog

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

https://app.datadoghq.com/monitors#downtime?id=410311687

Welcome, Chris! Get started You are 83% done setting up.

Triggered Monitors Manage Monitors Manage Downtime

Q Filter downtime

STATUS 1	SCOPE	MONITOR TAGS	MON
SCHEDULED	+	+	Issue {{host.ip}}
RECURRING	+	+	Issue {{host.ip}}
SCHEDULED	+	+	Issue {{host.ip}}
RECURRING	+	+	Issue {{host.ip}}
ACTIVE	+	+	One

Downtime

Scope: *

Monitor: Issue with metric: my_metric on host: {{host.ip}}

Group scope (optional, default all groups): *

Preview affected monitors

2 Schedule

One-Time Recurring

Start Date: 2018/10/27 Time Zone: America/New_York

Repeat Every: 7 days

Beginning: ..:..

Duration: 2 days

Repeat Until: No end date

Summary: From 9:00am to 9:00am in 2 days Every 7 days

Preview planned recurrences

3 Add a message

Preview Edit

Markdown supported

Please be advised that the monitor for my_metric will be silenced from 9am Saturday until 9am Monday. @creites@gmail.com

Contact us!



Section 4

COLLECTING APM DATA

Using the Datadog platform, it's very easy to instrument applications for performance monitoring, either by running the process with ddtrace-run or by explicitly instrumenting your application. In the below example, I've instrumented a Flask application to feed metrics related to the performance of several routes.



```
ubuntu@ip-172-31-8-44: ~/flask_scripts
File Edit View Search Terminal Help
ubuntu@ip-172-31-8-44:~/flask_scripts$ cat flask_app.py
#!/usr/bin/env python
from flask import Flask
from flask import request
from time import sleep

# Import the necessary modules for tracing our app
import blinker as _
from ddtrace import tracer
from ddtrace.contrib.flask import TraceMiddleware

import logging
import sys
import random
import requests

# 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__)

# Instrument our app
traced_app = TraceMiddleware(app, tracer, service="sample-flask-app", distributed_tracing=False)

@app.route('/')
def api_entry():
    return 'Entrypoint to the Application'

# Create a new method that does something interesting that we can measure
@app.route('/url_lookup')
def url_lookup():
    url_name= request.args.get("url-name")
    full_url= "http://" + url_name

    # Trace how much time this portion of the method takes
    with tracer.trace("setup_overhead", service = "url-lookup"):
        # Artificially insert some random overhead time to make things interesting
        random_nbr = float(random.randint(250,750))
        sleep (float(random_nbr/1000))

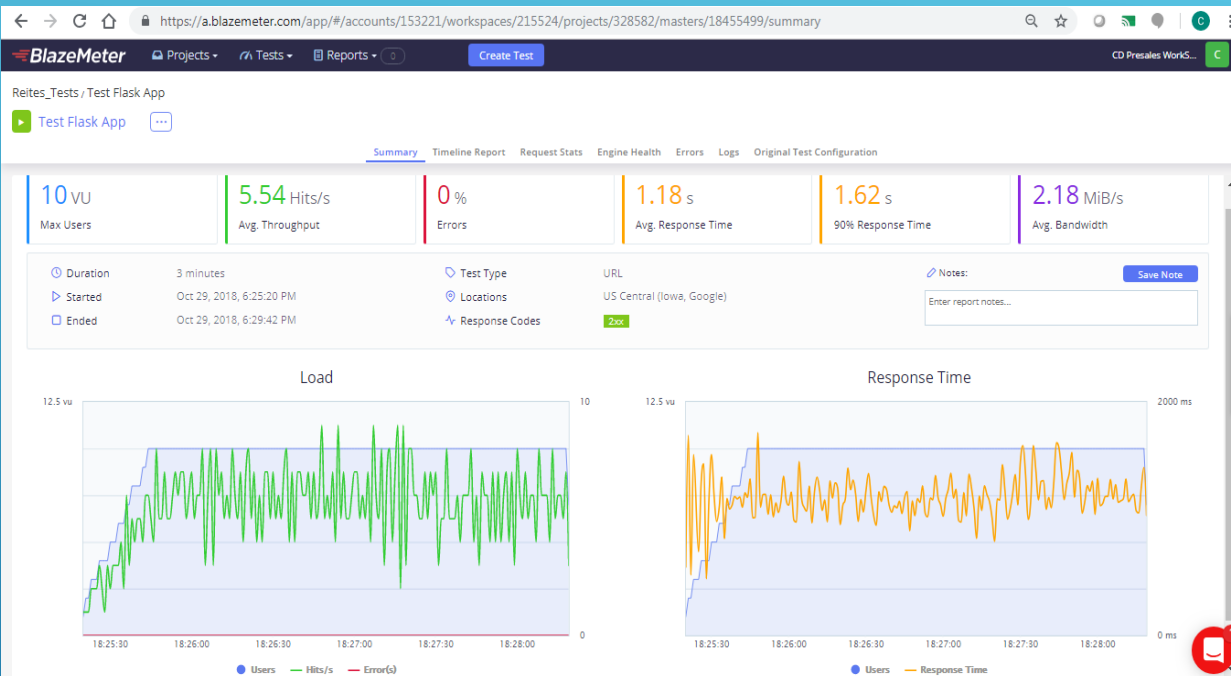
    # Trace how much time the actual URL call takes
    with tracer.trace("call_url") as span:
        # By setting a tag in the span, we can see which URL was being called when we look at our metrics
        span.set_tag("url-name", url_name)
        return requests.get(full_url).content

@app.route('/api/apm')
def apm_endpoint():
    return 'Getting APM Started'

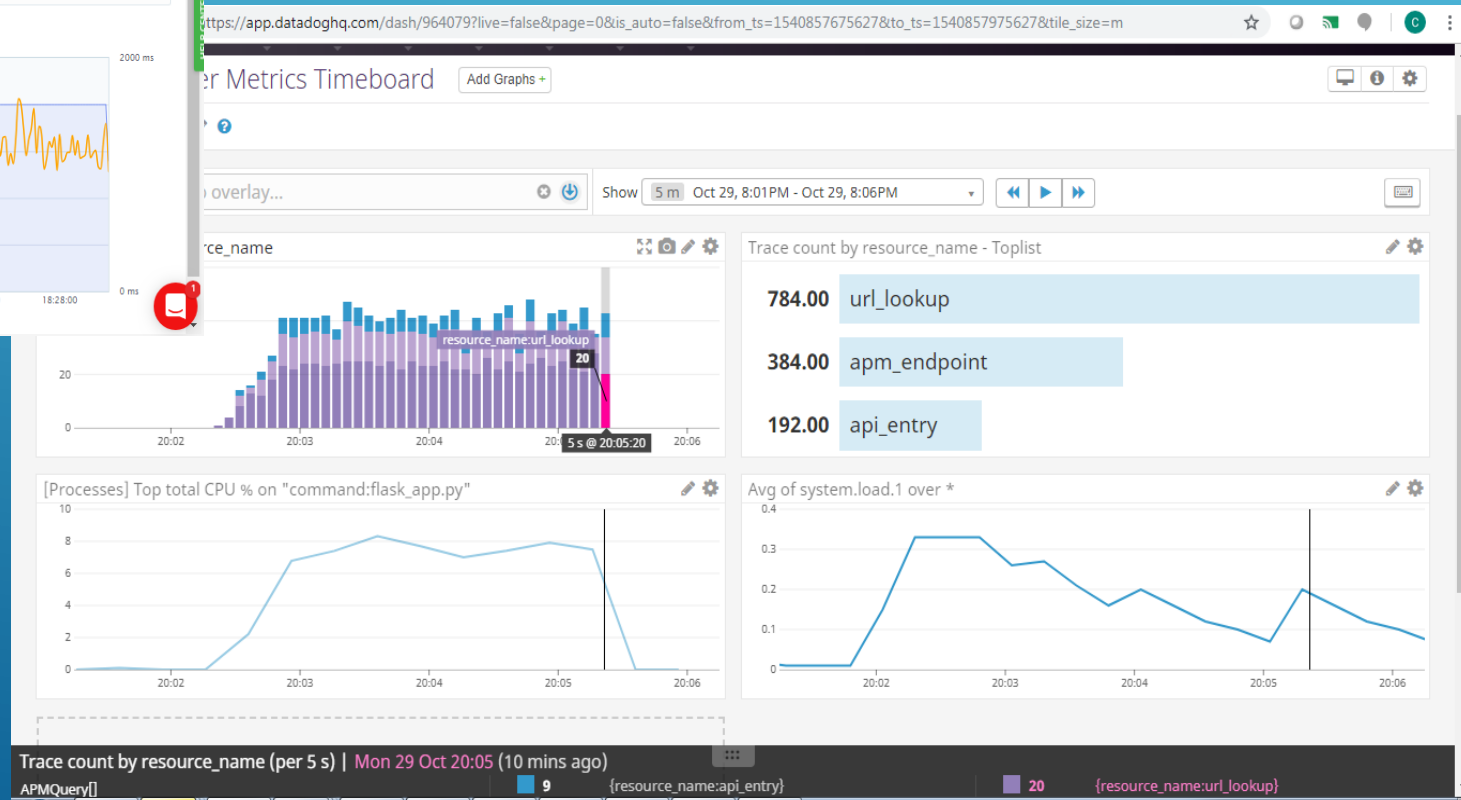
@app.route('/api/trace')
def trace_endpoint():
    return 'Posting Traces'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5050)
ubuntu@ip-172-31-8-44:~/flask_scripts$
ubuntu@ip-172-31-8-44:~/flask_scripts$
```

After I instrumented my application, I then used BlazeMeter, a cloud based load generation tool, in order to send thousands of requests to my application. The Timeboard below shows the counts by resource name, along with metrics about the CPU usage of the python app, and overall CPU utilization.

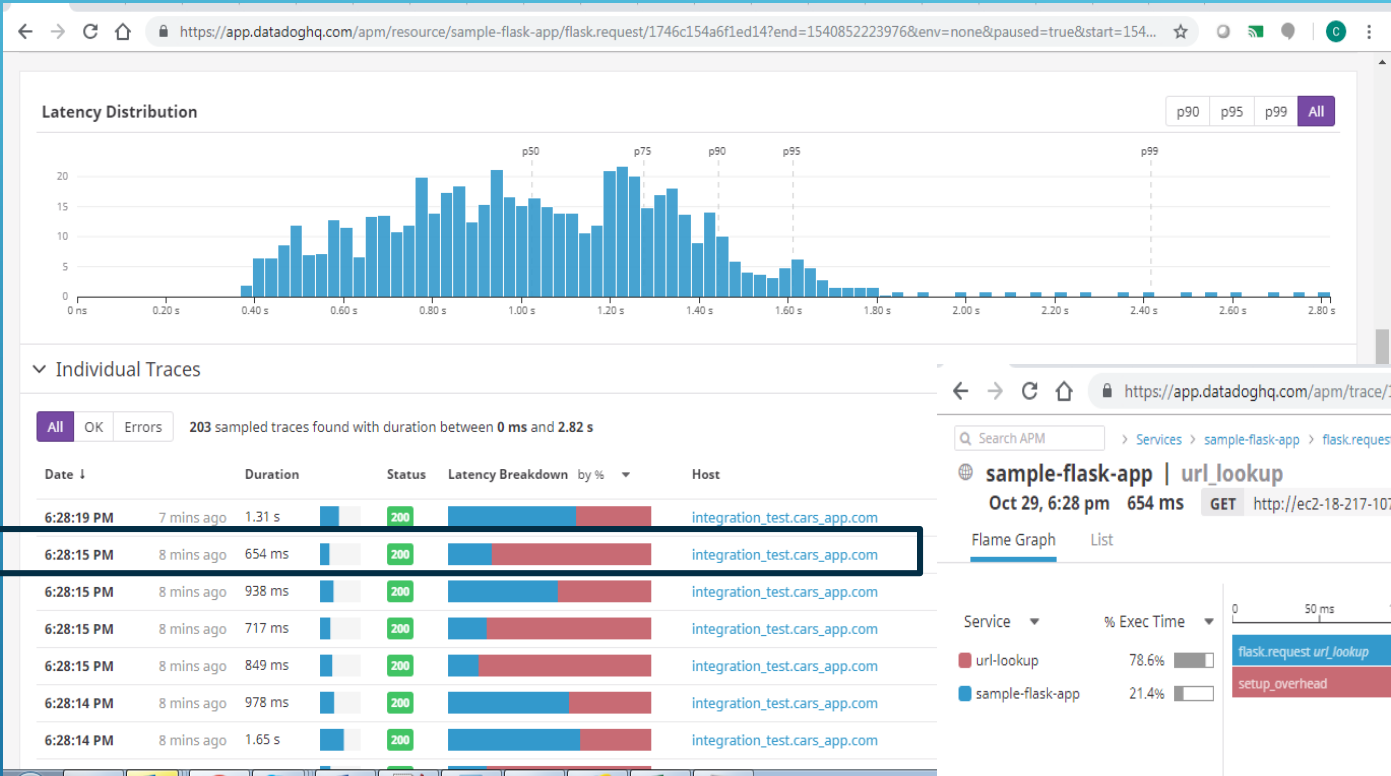


https://app.datadoghq.com/dash/964079?live=false&page=0&is_auto=false&from_ts=1540857675627&to_ts=1540857975627&tile_size=m&fullscreen=false

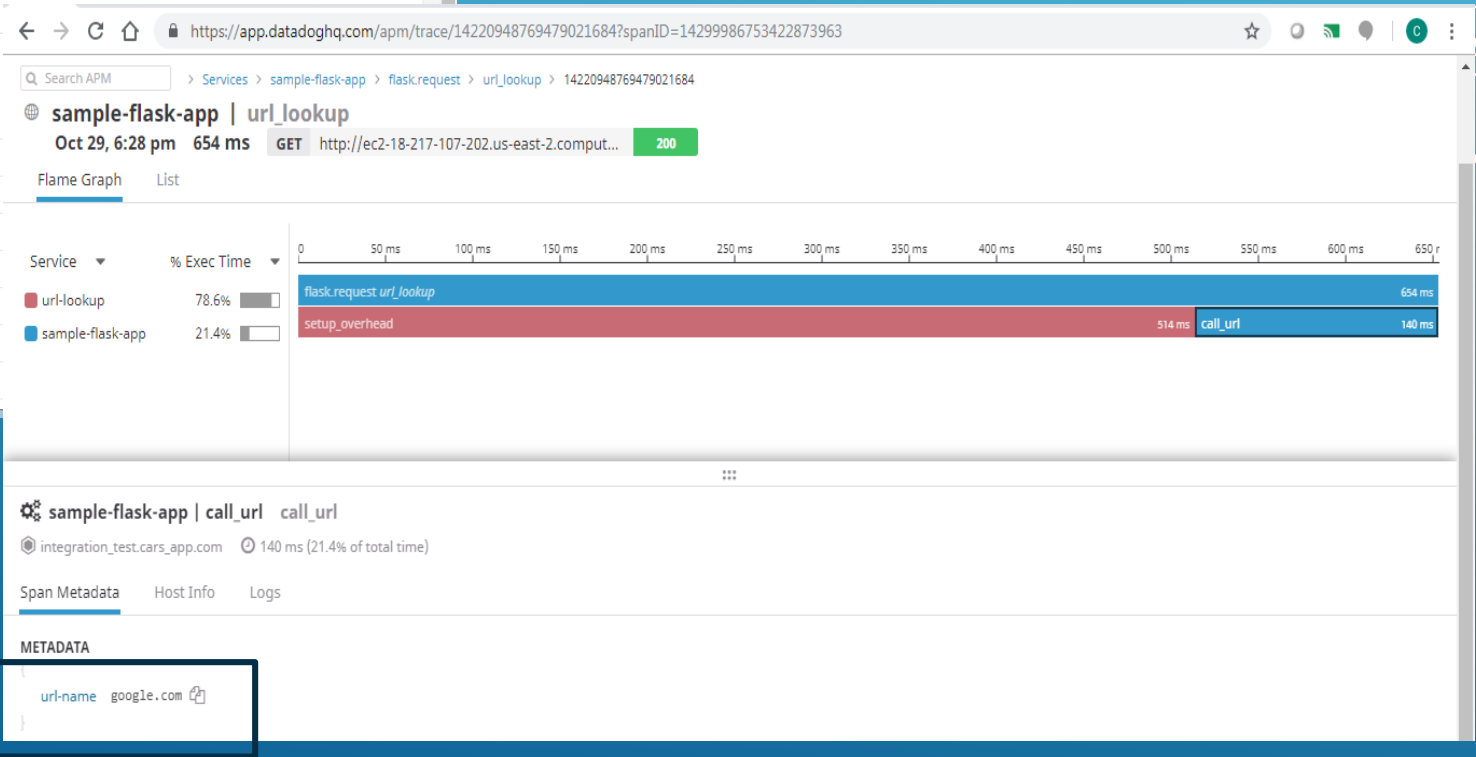




If we dive in a little deeper into our APM metrics, we can analyze latency as well. Because we setup a span with a tag name that matches the URL being loaded, we can also drill into individual traces and see the breakdown of time spent, along with the span data which shows the URL our app was trying to load on this particular call.



```
# Trace how much time the actual URL call takes
with tracer.trace("call_url") as span:
    # By setting a tag in the span, we can see which URL
    span.set_tag("url-name", url_name)
    return requests.get(full_url).content
```



Helpful Tip: What's the difference between a Service and a Resource?
A **resource** is one particular function which is typically part of a larger collection of other resources. Examples would be route functions of a web application.
A **service** is a collection of processes which together provides a set of functionality.



Section 5

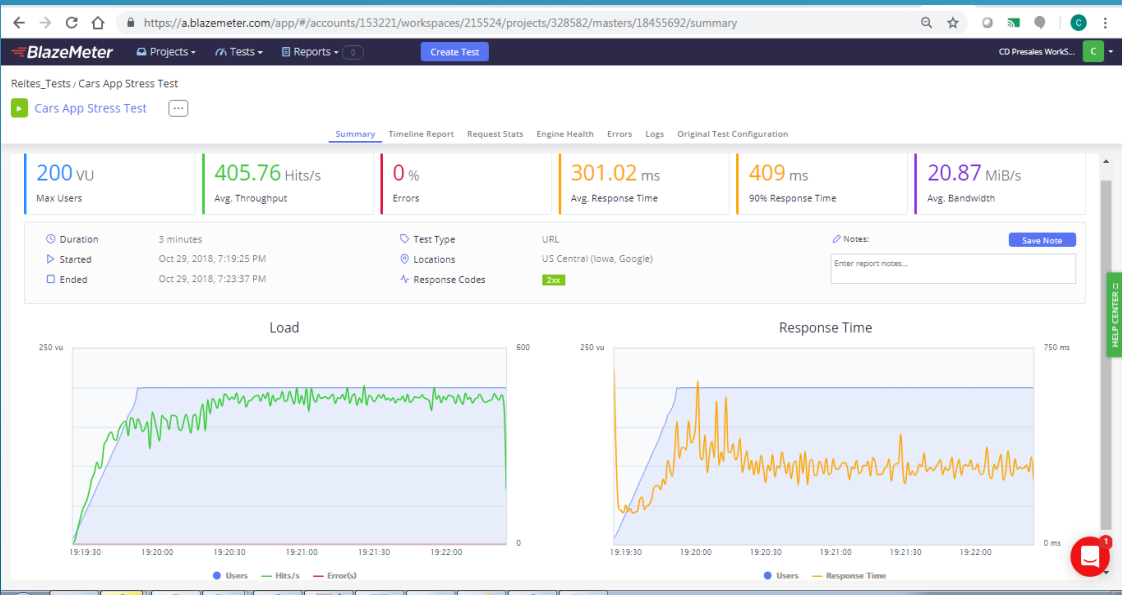
BUT WAIT... THERE'S MORE!!!



Prior to beginning this exercise, I had signed up for a free account on Datadog and was exploring functionality on my own. Because I had java applications, I installed the Java integration and ran some tests. The first thing I had to do was setup my application to accept JMX connections. I then had to configure the conf.yaml for the jmx integration to know how to connect. After that it was time to run some tests using BlazeMeter. I was most interested in two aspects of my application: The Garbage Collection metrics as well as the backend database performance.

```
if [ "$WEB" = "true" ]; then
    echo "Starting WEB"
    java -Dfile.encoding=UTF8 -Dcom.sun.management.jmxremote.port=7199 -Dcom.sun.management.jmxremote.ssl=false $WEB_AGENT -jar cars-app-10.1.0.war &
fi
```

```
instances:
- host: localhost
  port: 7199
  user: monitorRole
  password: DD_READONLY_JMX
# # If the agent needs to connect to the host
```



For this same application, I set up a monitor to make sure all 4 components of the application were always up and running. This also allowed me to play around with enabling the process check functionality.



```
ubuntu@ip-172-31-8-44: /etc/datadog-agent/conf.d/process.d
File Edit View Search Terminal Help

tags:
#
#   - env:staging
#   - cluster:big-data
#
thresholds:
#
#   critical if no sshd or more than 8 sshd are running
#   critical: [1, 7]
#   warning if 1, 2, 6, 7 sshd processes are running
#   warning: [3, 5]
#   ok if 3, 4, 5 processes are running
#
try_sudo: False

- name: cars_esb
  search_string: ['java -Dfile.encoding=UTF8 -jar cars-esb-10.1.0.war

- name: cars_web
  search_string: ['java -Dfile.encoding=UTF8 -jar cars-app-10.1.0.war
```

https://app.datadoghq.com/monitors/6811298

Welcome, Chris! Get started You are 83% done setting up. You have 4 days left in your trial. Upgrade

Monitor status since 7 minutes ago (30 Oct, 11:19:39 EDT) Unmute

One or More Cars Components are Down

Properties

Live Process
ID: 6811298
Created by Chris Reites

MUTING Muted on all scopes

TAGS

QUERY `processes('java').over('user:ubuntu').rollup('count').last('5m') < 4`

PROCESSES [View LIVE matching processes --](#)

MESSAGE Please Investigate and restart services as needed. @creites@gmail.com
1 notified Chris Reites

https://app.datadoghq.com/process?columns=host,process,user,cpu,memory,start&options=normalizeC

Welcome, Chris! Get started You are 83% done setting up.

Processes Containers

cars Filter by tag...

Hide Sidebar Showing 1 - 4 of 4 matching processes

Host

Search 1 host

☐ i-03e9b2708b65bf9d0

User

Search 1 user

☐ ubuntu

Env

Search 1 env

☐ integration_test

PROCESS

```
java -Dfile.encoding=UTF8 -Dcom.sun.m
java -Dfile.encoding=UTF8 -jar cars-loan-
java org.springframework.boot.loader.W
java -Dfile.encoding=UTF8 -jar cars-esb-1
```

java

HOST i-03e9b2708b65bf9d0 **USERNAME** ubuntu 14 minutes ago 3360

TAGS host:i-03e9b2708b65bf9d0 command:cars-app-10.1.0.war env:integration_test name:cars_app state_zombie:false

FULL COMMAND `java -Dfile.encoding=UTF8 -jar cars-esb-10.1.0.war`

Total CPU % 15min 0.7 % (0.06 %) RSS Memory 15min 2.3 GiB (2.3 GiB)

30
20
10
0

11:20 11:25 11:30

2
1
0

11:20 11:25 11:30



Section 6

FINAL QUESTION

IMPROVING THE EFFICIENCY OF A BIKE SHARING BUSINESS WITH DATADOG



After having used several of the features that make up the Datadog platform, I began thinking of a fun use case that Datadog could bring value to. Having lived a big portion of my life in Central Florida (Mickey Mouse Town), a lot of the things I thought about were theme park related (ride wait times, parking, trend analysis of concession sales, etc...) but then while riding my bike to lunch today, I thought of a great use case having to do with bike shares.

Bike share programs are a great solution to commuting woes in large cities, however they're often plagued by a common problem. As users rent bikes and transport them to wherever they're headed, you end up with too many bikes in some places, and not enough bikes where you need them.

By utilizing the Datadog platform, you could keep track of how many bikes were currently in each location, and based on monitoring criteria, you could notify the appropriate "bike relocation" staff to transport bikes from areas with low demand to areas with high demand. You could also use historical trend analysis to proactively move bikes into areas where you are forecasting upcoming higher demand.



THANK YOU



I really enjoyed getting to explore the Datadog platform over the past week. Datadog has a fantastic product from what I've seen. So many customers I talk with today are using several different tools from multiple vendors to accomplish what Datadog can do on a single cloud based platform. I would greatly value the chance to join the team there and help drive the Datadog business forward while enabling customers to solve the many challenges they have around infrastructure, application, and log monitoring. If you have any questions for me about this exercise, please feel free to reach out to me at creites@gmail.com.

Thank You,
Chris