DATADOG

SE Candidate Summary Report

Prepared by:

Will Zuill, Solution Engineer (SE) Candidate

Date:

11 December 2018

CONTENTS

Executive Summary

Overview

As part of Datadog's continued focus on ensuring the highest quality service and support to their customers they require all potential pre-sales solution engineer candidates to complete a technical exercise designed to test various aspects of their skills.

This exercise not only validates the candidates ability to problem solve and their technical expertise, but also demonstrates their ability to summarize the value of Datadog solutions covered by the various use cases the candidate performs.

This document details my findings and conclusions from the exercise. It is divided into two (2) main areas:

- The Main Summary
- The Appendix detailed the technical approach taken

Approach

As all industries today have an increased focus on time to market, agility, and digital transformation, it is critical for organizations to adopt strategies that will streamline the development and deployment of their business services.

Strategies such as 'DevOps' offer companies many benefits to achieve high agility but also come with challenges that organizations must overcome. One such challenge is the provisioning and maintaining the environments required for this fast-paced approach using traditional strategies can be time consuming.

This has created an industry shift towards 'containerization' allowing environments to be deployed in milliseconds whilst still maintaining concurrency.

Therefore, I elected to conduct the exercise using a containerized approach, as I strongly believe that this configuration will be encountered more and more by Datadog.

Collecting Metrics

Summary

Datadog's unique approach allows organizations to monitor all aspects of the IT infrastructure (cloud, on premise servers, hybrid etc.) in one single view. This allows teams, regardless of location, role to gain insight into the health of criteria business services and proactively collaborate and resolve issues before they occur.

In order for teams to fully understand their environment several simple steps should be taken. They are detailed below.

Exercise I – "Tagging"

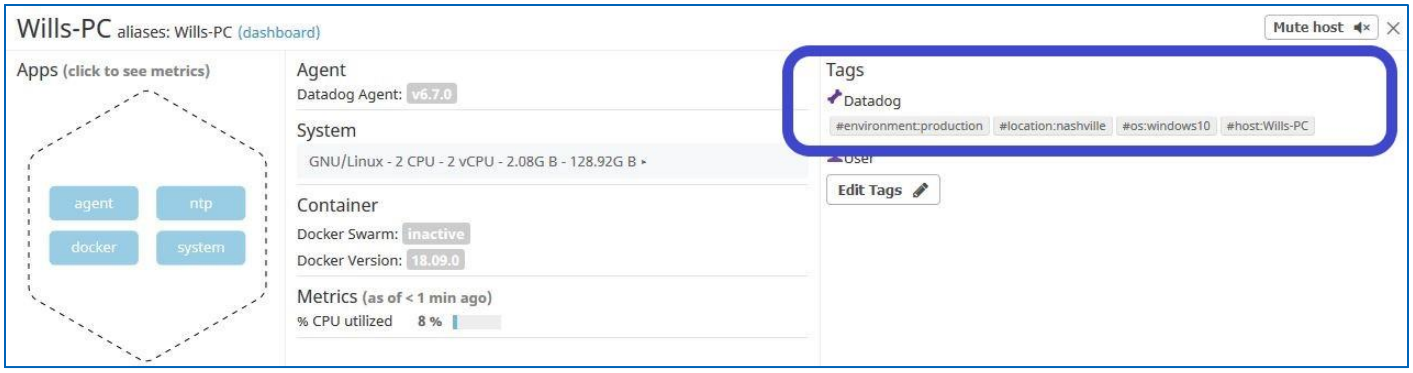
The first step to gaining a complete understanding of an organizations environment is to identify criterial information about each asset being monitored that is outside of the metrics detailed by the asset itself.

For example, an organization may wish to know all of the servers within a specific location so, should an issue arise, they can quickly determine if it's affecting a single component or all of the components in a location.

"Tagging" allows assets to be identified and grouped by a series of custom labels. These labels can then be filtered on within the Datadog interface within one click – allowing teams to sort information as required.

Figure 1.1 below details how I was able to add custom tags to an example asset within my organization with the type of environment, home city, OS and host name.

Figure 1.1



Wills-PC aliases: Wills-PC (dashboard)

Apps (click to see metrics)

Agent
Datadog Agent: v6.7.0

System
GNU/Linux - 2 CPU - 2 vCPU - 2.08G B - 128.92G B ▸

Container
Docker Swarm: inactive
Docker Version: 18.09.0

Metrics (as of < 1 min ago)
% CPU utilized 8 %

Tags
Datadog
#environment:production #location:nashville #os:windows10 #host:Wills-PC
User

Exercise II – Collect Metrics from External Source

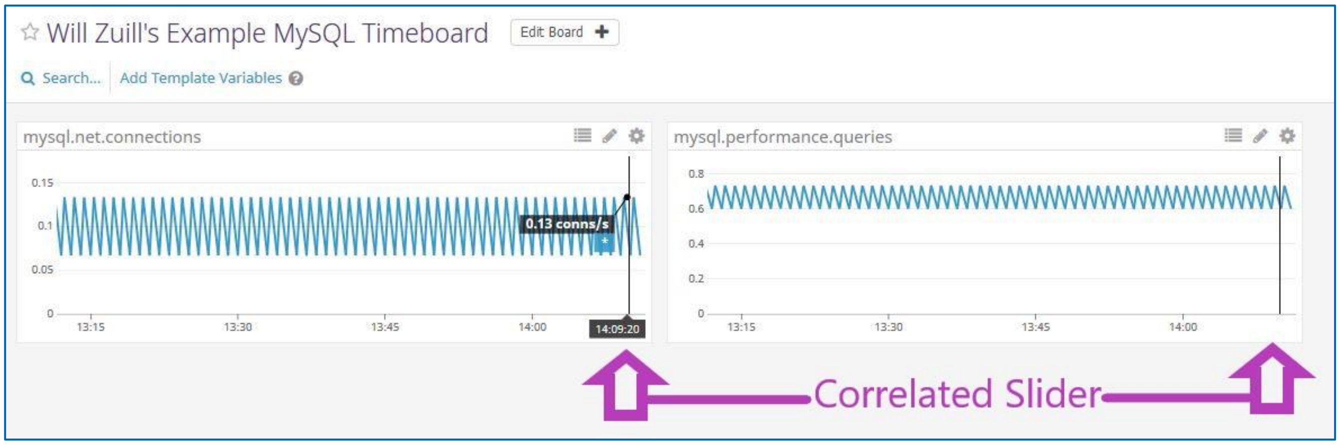
As mentioned above, the key to proactively monitoring (and therefore preventing problems before they occur) is the monitoring of all aspects of an organizations infrastructure in a single view.

This involves gathering criterial metrics from all of the systems, applications and services across your entire organization. Datadog provides more the two-hundred and fifty (250) turn key industry solutions, all fully supported. The full list is available here: https://www.datadoghq.com/product/integrations/

As an illustration of this, figure 1.2 details two (2) metrics from my local MySql installation visually in a dashboard. It should be noted that metrics from multiple related sources can be combine into a single 'Timeboard' which allows events to be correlated to help teams quickly determine root cause.

In figure 1.2, the arrows details a marker on each of the two graphs that can be precisely matched with a simple slide of a mouse

Figure 1.2



Exercise III – Create a Custom Check

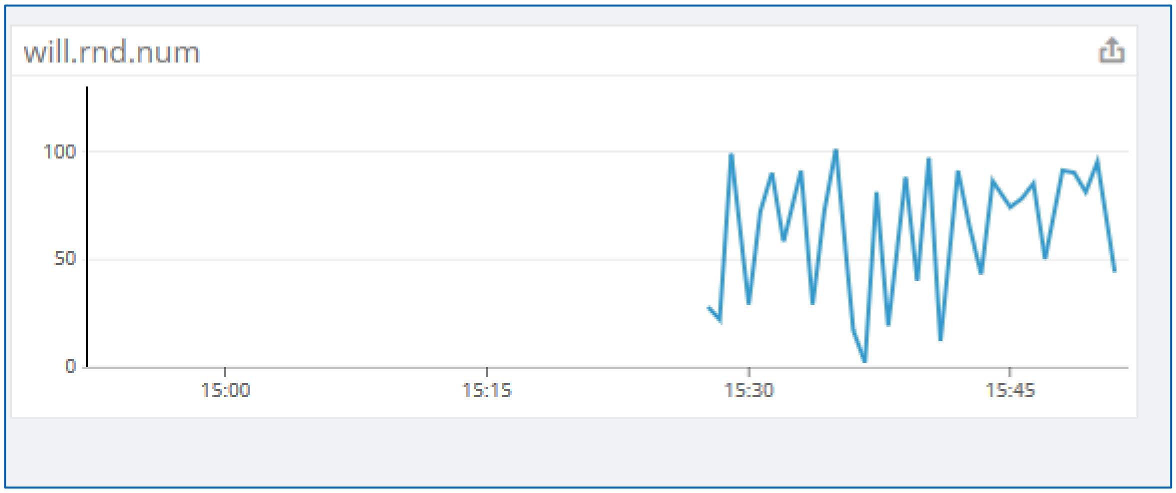
Whilst Datadog provides more than two-hundred and fifty (250) integrations with industry standard solutions it also provides the ability to create custom checks/integrations for solutions that are outside of the support list. For example, an organization has developed their own in-house monitoring solution.

Custom checks can gather metrics from such systems and present them to Datadog so they can be utilized as any other supported metrics/check. Of course, it is recommend that, where possible, a support check is used.

A custom check provides an extremely flexible way to gather such metrics and the intervals in which they are collected. For example, should a team wish to collect custom metrics at a different interval that the default fifteen (15) second they can do so be simple setting an option (min_collection_interval) in the configuration file rather than modify code. This allows teams without coding experience to control the collection rate.

As an example, I created a custom check that generates a random number between 1 and 100 every forty five (45) seconds. Figure 1.3 details how this custom check (will.rnd.num) can then be visualized in Datadog over time.

Figure 1.3



Exercise I – Create TimeSeries Dashboard Using API

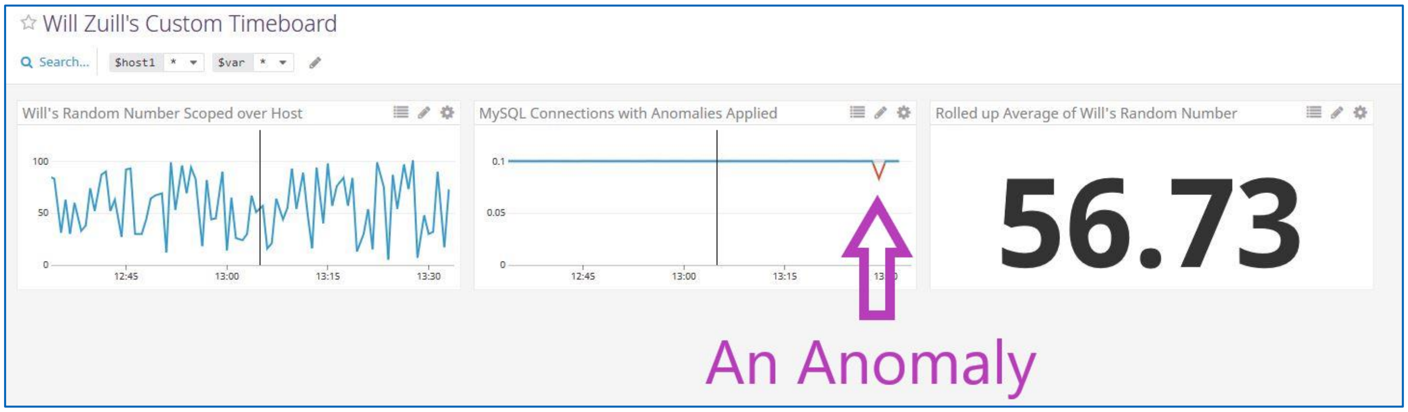
Datadog provides multiple ways to visualize data within a Dashboard format. For example, users can create compelling dashboards using Datadog's intuitive user interface in a matter of minutes using drag-and-drop.

Users are also able to utilize Datadog's extensive API to create 'custom' dashboard to meet their needs. In my example I created a custom dashboard using the API function (see script in appendix) to visualize three (3) metrics (see figure 2.1):

1. A Custom Metric
2. A metric from MySql configured to visually alert on any anomalies (see below)
3. A rolled up, custom metric

As mentioned above, the middle graph on the dashboard below has been customized to visually alert on anomalies. Anomalies are metrics that are behaving differently from the 'standard' value it's had in the past. This means the metric is averaged over a user defined period of time (for example, one week). Then if the metric is different (by a user defined threshold) from that established threshold, it will be visually identified on the metrics graph (i.e. on a line graph the line will turn red is the anomaly is detected. This process can also be used to send alerts to the team.

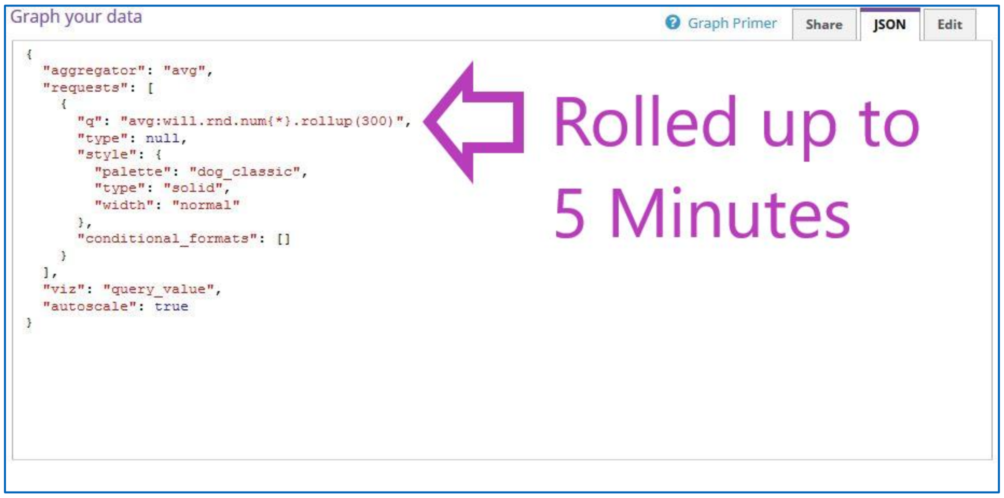
Figure 2.1


Will Zuill's Custom Timeboard
Search...
$host1 *
$var *
Will's Random Number Scoped over Host
100
50
0
12:45
13:00
13:15
13:30
MySQL Connections with Anomalies Applied
0.1
0.05
0
12:45
13:00
13:15
Rolled up Average of Will's Random Number
56.73
An Anomaly

Exercise II – Change Timescale and Share

Once a graph has been added to a dashboard, it can be simply be customized to change its timescale from the other graphs on the dashboard. For example, if you wished to 'rollup' a counter to a five (5) minute interval (defined in seconds i.e. 300 seconds), the rollup function can be applied (see figure 2.2)

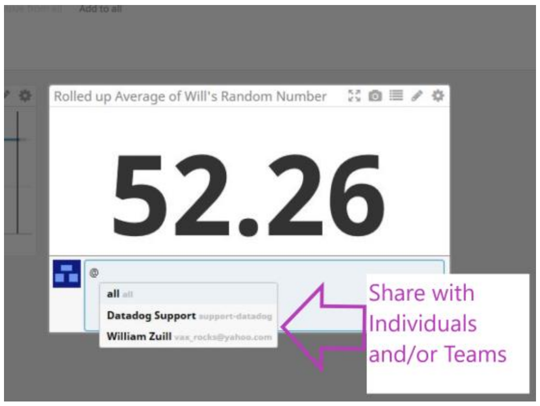
Figure 2.2



```
{
  "aggregator": "avg",
  "requests": [
    {
      "q": "avg:will.rnd.num{*}.rollup(300)",
      "type": null,
      "style": {
        "palette": "dog_classic",
        "type": "solid",
        "width": "normal"
      },
      "conditional_formats": []
    }
  ],
  "viz": "query_value",
  "autoscale": true
}
```

Any graph on a dashboard can also be shared with individual or teams as a snapshot. This can be particularly useful when teams collaborating on issues. As example, I took a snapshot of one of the graphs and shared it with myself (see figure 2.3).

Figure 2.3


Add to all
Rolled up Average of Will's Random Number
52.26
@
all all
Datadog Support support-datadog
William Zuill vax_rocks@yahoo.com
Share with
Individuals
and/or Teams

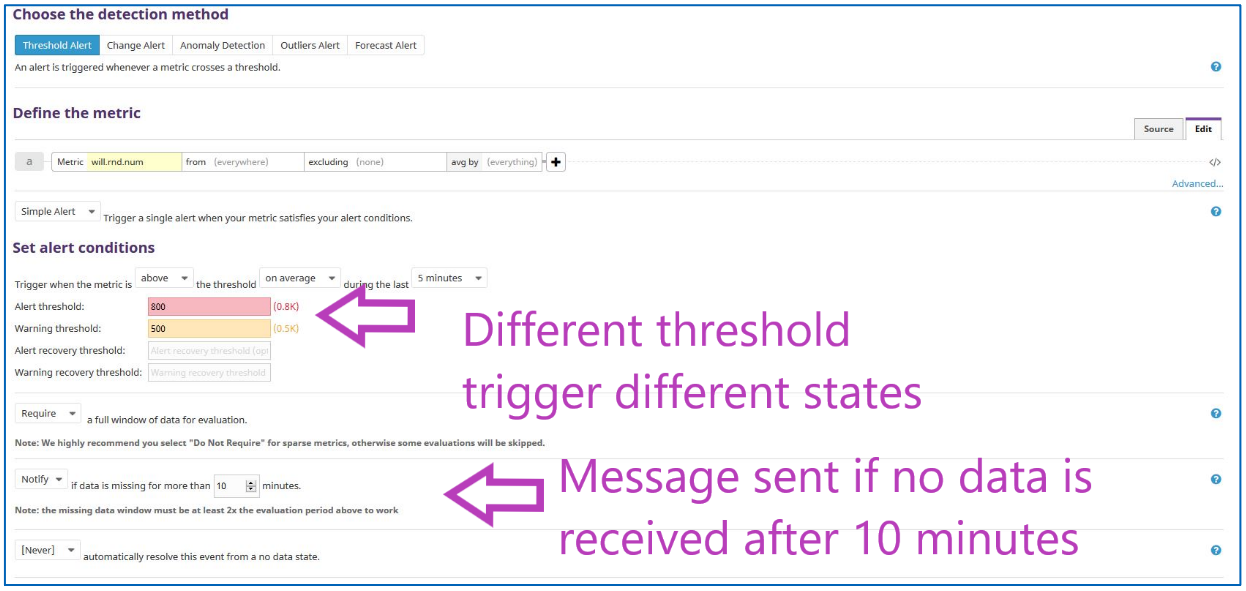
Exercise I – Create Monitor

As mentioned in the previous section, it is important that teams/individuals are informed if key metrics are indicating that there is a problem or, more proactively, if they are trending towards a problem.

Datadog provides the ability to monitor critical metrics and send custom alerts depending on the value of the metrics. In the following example, a metric sends a value of between 0-1000. The monitor is created to track the desired metric (See figure 3.1) and send a different set of messages depending on if it's either:

1. Not sending any data after ten (10) minutes
2. In a warning state if the value of the metric is above 500
3. In an alert state if the value of the metric is above 800

Figure 3.1



Choose the detection method

An alert is triggered whenever a metric crosses a threshold.

Define the metric

a | Metric will.rnd.num | from (everywhere) | excluding (none) | avg by (everything)

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

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 (opt	
Warning recovery threshold:	Warning recovery threshold	

Different threshold trigger different states

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

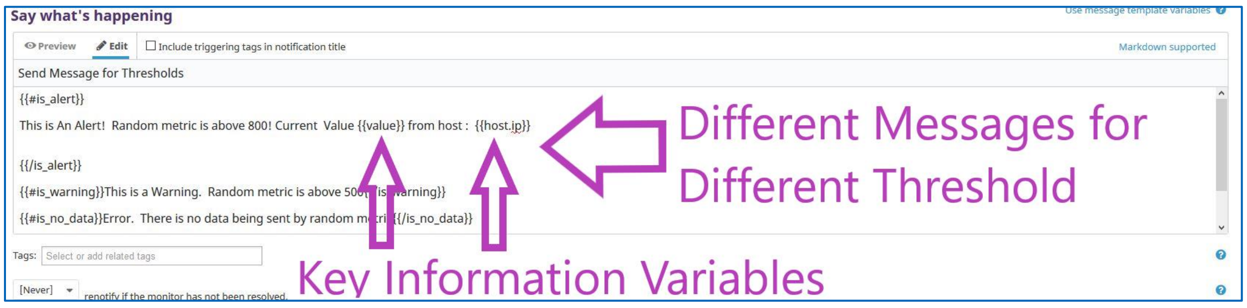
Message sent if no data is received after 10 minutes

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

Exercise I – Create Monitor

Datadog also provides the ability to send different messages depending on the type of alert. For example, a unique message is generated for warnings, alerts and no data. This can be accomplished through a very simple message. You can also include critical information (Such as the IP address of the machine that is experiencing the issue) through simple variables (See Figure 3.2)

Figure 3.2


Say what's happening
Use message template variables
Preview
Edit
Markdown supported
Different Messages for Different Threshold
Key Information Variables

☐ Include triggering tags in notification title

Send Message for Thresholds

```
{{#is_alert}}
This is An Alert!  Random metric is above 800! Current  Value {{value}} from host :  {{host.ip}}

{{/is_alert}}
{{#is_warning}}This is a Warning.  Random metric is above 500{{/is_warning}}
{{#is_no_data}}Error.  There is no data being sent by random metric{{/is_no_data}}
```

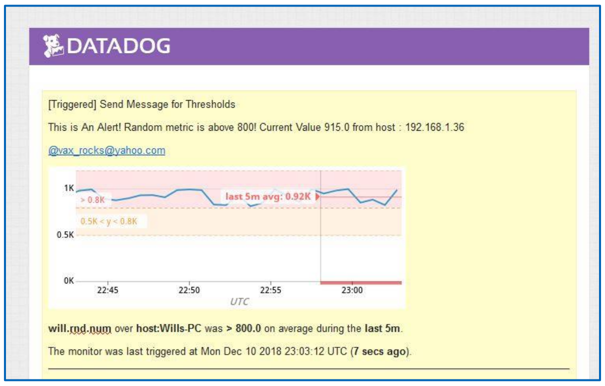
Tags: Select or add related tags

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

This ability to create different messages for different types of alerts allows teams/individuals to place appropriate emphasis on any issue depending on the severity of the alert. It also allows teams/individuals to be proactive i.e. being the resolution of a problem as it's trending towards an issue but **before** it causes distribution.

An example of the resulting email is detailed in figure 3.3 below.

Figure 3.3



[Triggered] Send Message for Thresholds

This is An Alert! Random metric is above 800! Current Value 915.0 from host : 192.168.1.36

@vax_rocks@yahoo.com

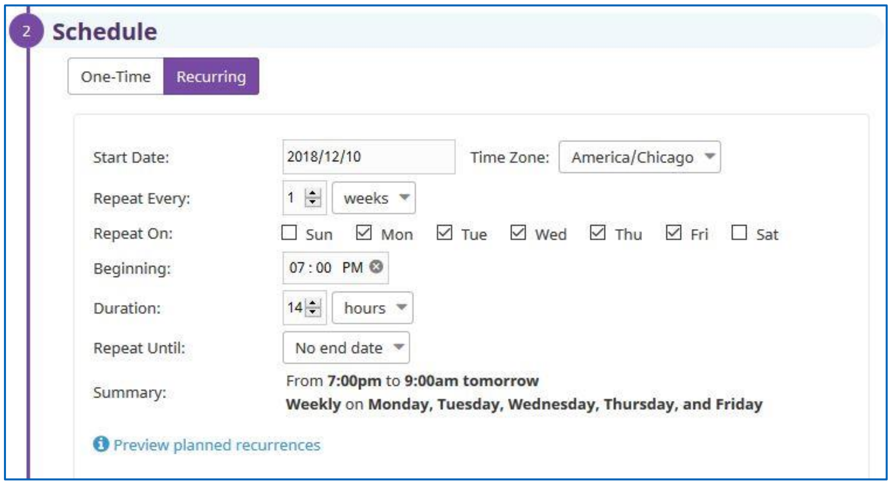
will.rnd.num over **host:Wills-PC** was **> 800.0** on average during the **last 5m**.

The monitor was last triggered at Mon Dec 10 2018 23:03:12 UTC (**7 secs ago**).

Exercise II – Manage Downtime

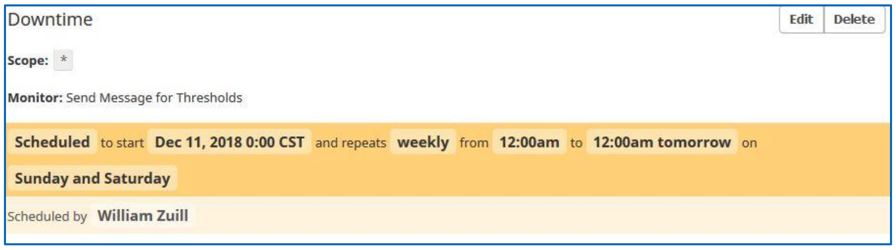
As it is possible that it is not required that a monitored metric sends a message at certain times, Datadog provides the ability to specify 'downtime'. For example, if you do not wish to send a certain alert to team members outside of core hours, then you can easily specify this within the Datadog user interface. See Figure 3.3 specifying that this monitor will not send messages between 5pm and 9am Monday-Friday. See Figure 3.4 detailing that the monitor will not send messages during the weekend.

Figure 3.3



Start Date:	2018/12/10 Time Zone: America/Chicago
Repeat Every:	1 weeks
Repeat On:	☐ Sun ☑ Mon ☑ Tue ☑ Wed ☑ Thu ☑ Fri ☐ Sat
Beginning:	07 : 00 PM
Duration:	14 hours
Repeat Until:	No end date
Summary:	From **7:00pm** to **9:00am tomorrow** **Weekly** on **Monday, Tuesday, Wednesday, Thursday, and Friday**

Figure 3.4



Downtime

Scope: *

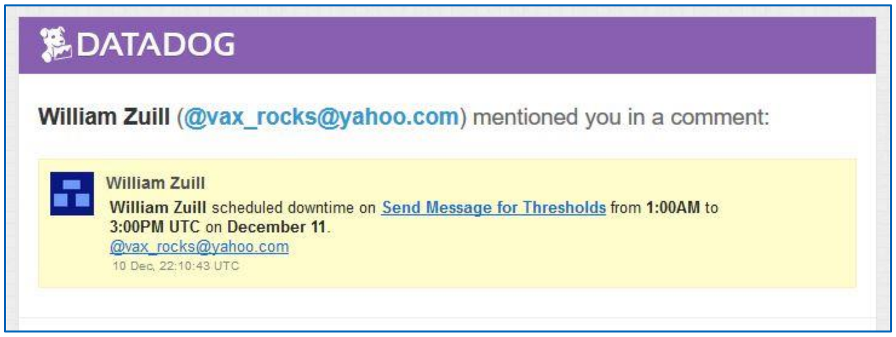
Monitor: Send Message for Thresholds

Scheduled to start **Dec 11, 2018 0:00 CST** and repeats **weekly** from **12:00am** to **12:00am tomorrow** on **Sunday and Saturday**

Scheduled by **William Zuill**

Users are kept informed should a monitor change via email. An example of the email is detailed in figure 3.5

Figure 3.5



William Zuill (@vax_rocks@yahoo.com) mentioned you in a comment:

William Zuill

William Zuill scheduled downtime on Send Message for Thresholds from **1:00AM** to **3:00PM UTC** on **December 11**.

@vax_rocks@yahoo.com

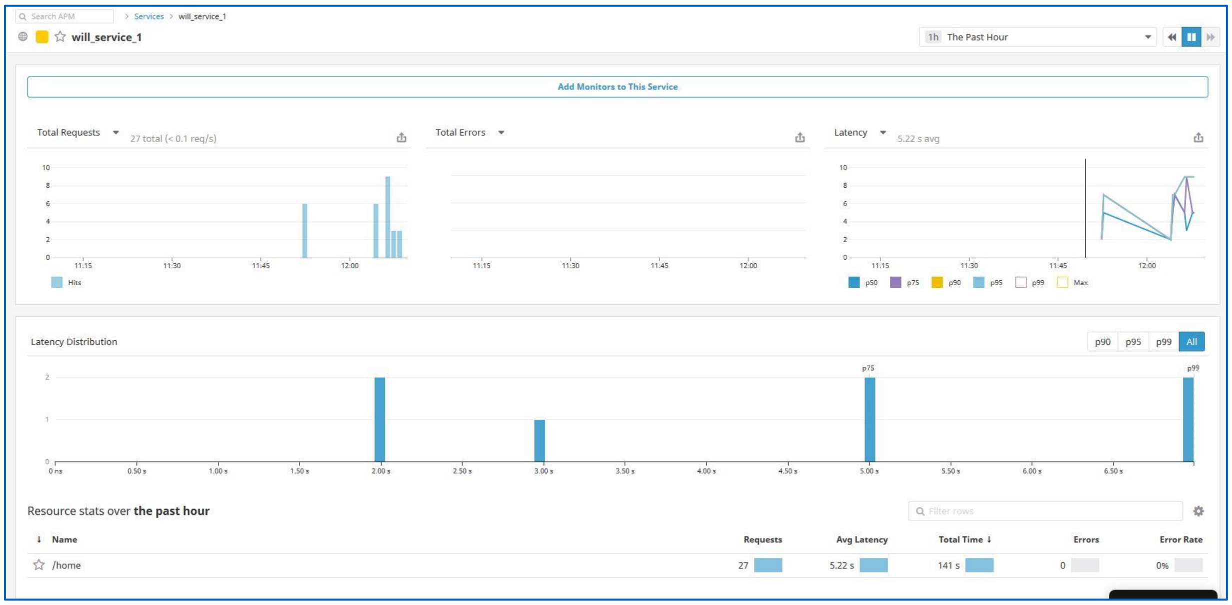
10 Dec, 22:10:43 UTC

Exercise I – Instrument Application with APM

In order to gain a complete understanding of an environment it is important to not only measure and monitor the infrastructure but also the business services (or applications) themselves. By correlating information on what the application is doing at a given time with infrastructure metrics, teams are able to see how individual operations within a business service affect or are affected by infrastructure. For example, how does an individual database request from an application affect the overall database server? This allows teams to not only quickly and easily troubleshoot issues but also develop more robust applications, faster.

Within Datadog, the ability is provided to trace each action within an application and then visually the breakdown of each action within the Datadog UI. Applications written in multiple languages can be instrumented for Datadog in sixty (60) seconds or less and then, using the dashboard technology previously discussed, visualized as required. Figure 4.1 details an example dashboard showing application information and figure 4.2 shows an example dashboard that marries infrastructure data with information from the application itself.

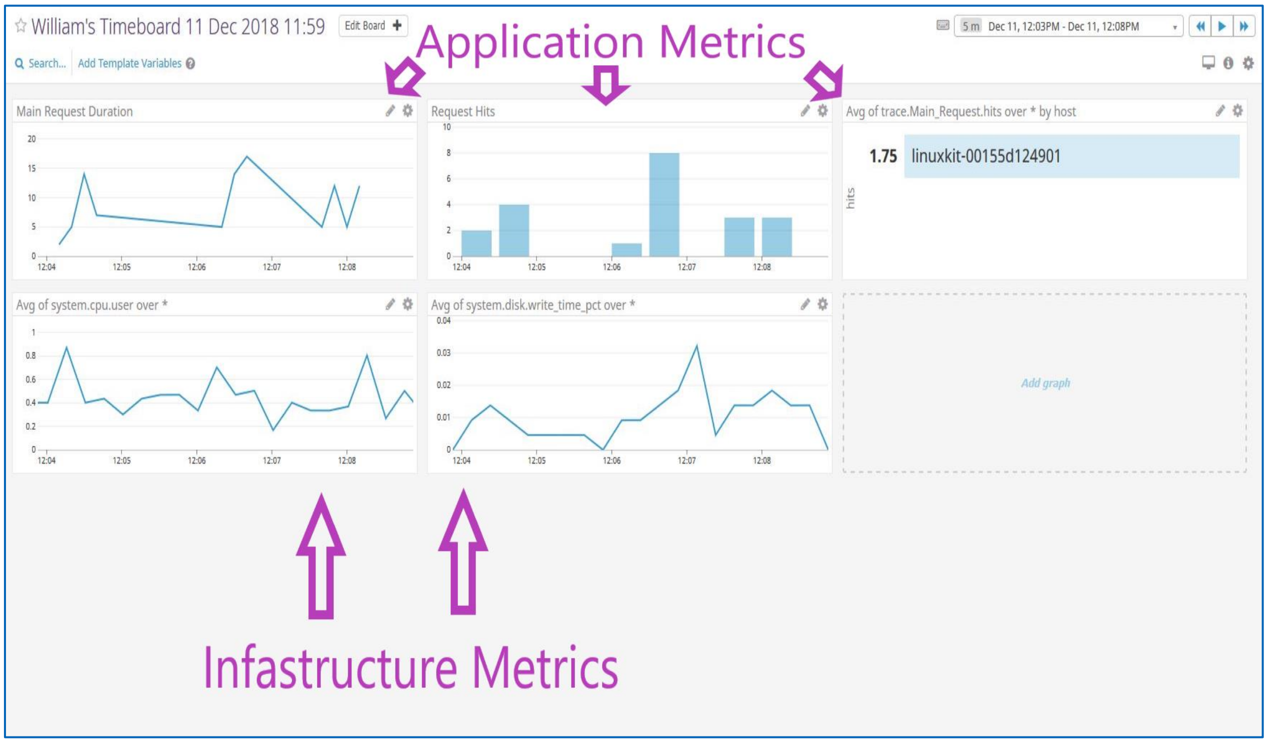
Figure 4.1



Resource stats over **the past hour**

↓ Name	Requests	Avg Latency	Total Time ↓	Errors	Error Rate
/home	27	5.22 s	141 s	0	0%

Figure 4.2


William's Timeboard 11 Dec 2018 11:59
Edit Board
Search...
Add Template Variables
5 m Dec 11, 12:03PM - Dec 11, 12:08PM
Application Metrics
Main Request Duration
Request Hits
Avg of trace.Main_Request.hits over * by host
1.75 linuxkit-00155d124901
hits
Avg of system.cpu.user over *
Avg of system.disk.write_time_pct over *
Add graph
Infastructure Metrics

Summary

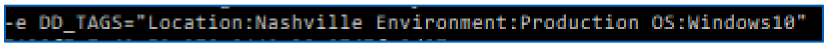
In order to keep pace and retain a competitive advantage in today’s digital market place, organizations have to focus on delivering and monitoring reliable and error free business services to its customers.

Datadog’s comprehensive solutions allows organizations to gain complete visibility into their business services. Datadog provides insight into all aspects that compose a business services from the infrastructure and to the application itself. By providing real-time monitoring and alerting, organizations can take a proactive approach in circumventing issues before they arise.

With more than 250 pre-built, supported integrations into 3rd party solutions that are common across the market, Datadog is the solution to ensure your business delivers its services at velocity.

Appendix

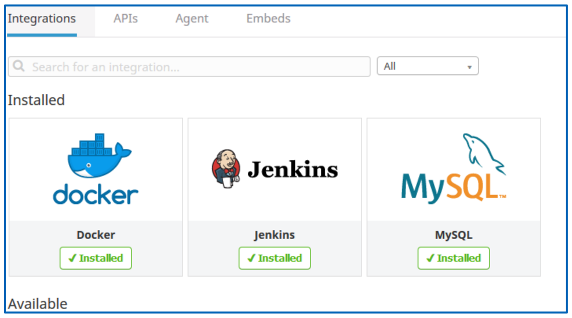
I – Tagging



```
-e DD_TAGS="Location:Nashville Environment:Production OS:Windows10"
```

Tags are achieved as through an environment variable as containers are initialized using ‘docker run’.

II – Collect Metrics from External Source



Integration is detailed via the integration menu option. Simply select the desired integration and follow the instructions.

III – Create Custom Check

Custom checks are stored in the ‘/etc/datadog-agent/checks.d’ directory and mush have a matching yaml in the /etc/datadog-agent directory. Once the agent is restarted, the metric will

IV – Create TimeSeries Dashboard Using API

The following is the code used to generate a TimeSeries Dashboard from an API.

```
options = {
    'api_key': '<HIDDEN>',
    'app_key': '<HIDDEN>'
}

initialize(**options)

title = "Will Zuill's Custom Timeboard"
description = "An Example Custom Dashboard"
graphs = [{
    "definition": {
        "events": [],
        "requests": [
            {"q": "avg:will.rnd.num{host:Wills-PC}"}
        ],
        "viz": "timeseries"
    },
    "title": "Will's Random Number Scoped over Host"},
{
    "definition": {
        "events": [],
        "requests": [
            {"q": "anomalies(avg:mysql.net.connections{*}, 'basic', 2, direction='both', alert_window='last_15m', interval=60,
count_default_zero='true')"}
        ],
        "viz": "timeseries"
    },
    "title": "MySQL Connections with Anomalies Applied"},
{
    "definition": {
        "events": [],
        "requests": [
            {"q": "avg:will.rnd.num{*}"}
        ],
        "viz": "query_value",
        "aggregator": "avg"
    },
    "title": "Rolled up Average of Will's Random Number"
}]

template_variables = [{
    "name": "host1",
    "prefix": "host",
    "default": "host:my-host"
}]

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

V – Change Timescale and Share

Timescales less than 1 hour can be changed directly in the JSON code of the graph/metric on the timeboard. Sharing is achieved through the snapshot icon on the graph.

VI – Create Monitor

Monitors are created directly from the Datadog UI. Each section within the monitor's creation offers a greater degree of control.

VII – Instrument Application with APM

The following is the custom application code used that was instrumented using the ddtrace command (for docker the environment variable DD_APM_ENABLED should be set to 'true').

```
rom flask import Flask, render_template


import logging
import sys
import time
import requests
import os
import urllib.request
import random
import json


# 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__)
```

Appendix

```
@app.route('/')
def api_entry():
# Create IDs.
   TRACE_ID = random.randint(1,1000000)
   SPAN_ID = random.randint(1,1000000)

# Start a timer.
# START and DURATION must be nanoseconds
   START = int(time.time() * 1000000000)

# Do things...
   time.sleep(5)

# Stop the timer.
   DURATION= int(time.time() * 1000000000) - START

# Send the traces.
   headers = {"Content-Type": "application/json"}

   data = [[{"trace_id": TRACE_ID, "span_id": SPAN_ID, "name": "Main Request", "resource": "/home",
"service": "Will Service 1", "type": "web", "start": START, "duration": DURATION}]]

   requests.put("http://localhost:8126/v0.3/traces", data=json.dumps(data), headers=headers)
# Create IDs.
   TRACE_ID = random.randint(1,1000000)
   SPAN_ID = random.randint(1,1000000)


# Start a timer.
# START and DURATION must be nanoseconds
   START = int(time.time() * 1000000000)

# Do things...
   time.sleep(9)

# Stop the timer.
   DURATION= int(time.time() * 1000000000) - START


# Send the traces.
   headers = {"Content-Type": "application/json”}
```

```
    data = [[{"trace_id": TRACE_ID, "span_id": SPAN_ID, "name": "Main Request", "resource": "/home",
"service": "Will Service 1", "type": "web", "start": START, "duration": DURATION}]]

    requests.put("http://localhost:8126/v0.3/traces", data=json.dumps(data), headers=headers)
# Create IDs.
    TRACE_ID = random.randint(1,1000000)
    SPAN_ID = random.randint(1,1000000)

# Start a timer.
# START and DURATION must be nanoseconds
    START = int(time.time() * 1000000000)

# Do things...
    time.sleep(3)

# Stop the timer.
    DURATION= int(time.time() * 1000000000) - START

# Send the traces.
    headers = {"Content-Type": "application/json"}

    data = [[{"trace_id": TRACE_ID, "span_id": SPAN_ID, "name": "Main Request", "resource": "/home",
"service": "Will Service 1", "type": "web", "start": START, "duration": DURATION}]]

    requests.put("http://localhost:8126/v0.3/traces", data=json.dumps(data), headers=headers)


     return 'Done!'

@app.route('/api/apm')
def apm_endpoint():
    time.sleep(10)
    return 'Getting APM Started'

@app.route('/api/trace')
def trace_endpoint():
    time.sleep(10)
    return 'Posting Traces'

if __name__ == '__main__':
    time.sleep(10)
     app.run(host='0.0.0.0', port='5050')
```