## Prerequisites - Setup the environment

For the exercise I downloaded Vagrant and installed a Vagrant Ubuntu VM with VirtualBox on macOS Mojave Version 10.14.6.

For instructions on how to download and install Vagrant refer to the following link: https://learn.hashicorp.com/collections/vagrant/getting-started

After downloading the Vagrant installer, vagrant is automatically added to your system path so that it is available in your terminal.

Initialize Vagrant:

“vagrant init hashicorp/bionic64”

Start the virtual machine:

“vagrant up”

Verify the installation worked by checking vagrant is available:

“vagrant status”

Datadog can be installed on Ubuntu using a one-step install:

“DD\_AGENT\_MAJOR\_VERSION=7 DD\_API\_KEY=<removed> DD\_SITE="datadoghq.com" bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install\_script.sh)"

Sign up for Datadog:

Use “Datadog Recruiting Candidate” in the “Company” field.

## 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.**

Added the following code to datadog.yaml:

"

tags:

- environment:dev

- env:ubuntu

- version:lukes-vm

"

After updating any code file inside the Vagrant Agent, I would logout and run the following code to restart the Agent:

"vagrant reload"

Wait a couple minutes and then refresh your Host Map in Datadog. Click on your Vagrant Agent in the Host Map, the tags added to the datadog.yaml are now displayed on the right-hand side.

Graphical user interface, application

Description automatically generated

I initially used Docker since I have prior experience using it to spin up instances in the past. However, I switch to Vagrant because I ran into issues when adding tags. I created a configuration file, datadog.yaml, from the .example file in my Docker Agent, but didn’t see the tags on the Host Map. I also added tags as environment variables at container creation, but I didn’t see those tags added to the Host Map either. I then saw some, but not all, of the tags on the Host Map after waiting about 5-10 minutes. I tried to figure out where they were coming from by adding new, different tags to the configuration file and as environment variables at container creation, but I did not see these new tags after copying in the updated configuration file into Agent from my local machine and restarting the Agent. I did not run into any issues when using Vagrant.

https://media.giphy.com/media/UX08QMbe9BECjRoq3E/giphy.gif

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

For this part I chose to use Postgres because I have prior experience using it.

For instructions on how to download and install Postgres refer to the following link:

https://postgresapp.com/

I used Homebrew to install Postgres. I initially ran into integration issues with Postgres because I left the admin password blank when installed it. I found in the Postgres docs that doing this will make the admin password always fail. I resolved the issue by running Postgres as a super user to override default password.

https://media.giphy.com/media/JWF7fOo3XyLgA/giphy.gif

"sudo su - postgres"

"psql postgres"

Graphical user interface, application, website

Description automatically generated

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

I created the following files for my custom Agent check:

"checks.d/custom\_check.py"

"conf.d/custom\_check.yaml"

I added the following code to custom\_check.py to make the custom check compatible with any Agent version.

"

try:

from datadog\_checks.base import AgentCheck

except ImportError:

from checks import AgentCheck

from random import randint

\_\_version\_\_ = "7.23.1"

class HelloCheck(AgentCheck):

def check(self, instance):

self.gauge('my\_metric', randint(1,1000), tags=['check:check-randint'])

"

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

I changed my check's collection interval by adding the following code to custom\_check.yaml.

"

init\_config:

instances:

- min\_collection\_interval: 45

"

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

Yes, the collection interval can also be modified under "instances" in the configuration .yaml file.

**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**

I used the following code in a Postman POST request to create my Timeboard:

"

{

"title": "Luke's Timeboard",

"widgets": [

{

"definition": {

"type": "timeseries",

"requests": [

{

"q": "avg:my\_metric{\*}"

}

],

"title": "Average of My Metric Over Time"

}

},

{

"definition": {

"type": "timeseries",

"requests": [

{

"q": "anomalies(avg:postgresql.postgresql.percent\_usage\_connections{\*}, 'basic', '2')"

}

],

"title": "Postgres Percent Usage Anomalies"

}

},

{

"definition": {

"type": "timeseries",

"requests": [

{

"q": "avg:my\_metric{\*}.rollup(sum, 3600)",

"display\_type": "bars",

"style": {

"palette": "dog\_classic",

"line\_type": "solid",

"line\_width": "normal"

}

}

],

"title": "Hourly Rollup Sum of My Metric"

}

}

],

"layout\_type": "ordered",

"description": "This is my dashboard",

"is\_read\_only": **true**,

"notify\_list": [

"luke7@vt.edu"

],

"template\_variables": [

{

"name": "host",

"prefix": "host",

"default": "Luke's Host"

}

],

"template\_variable\_presets": [

{

"name": "Saved views for hostname 2",

"template\_variables": [

{

"name": "host",

"value": "Luke's second host"

}

]

}

]

}

"

I tried running the code in a Python script using the datadog library and the initialize and api modules, as shown in the documentation here:

https://docs.datadoghq.com/api/v1/dashboards/

However, after installing the necessary packages in the vagrant and running the script, I did not see my Timeboard created in the Dashboard List. I couldn't find any other documentation to resolve the issue, so I sent the request from Postman.

**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.**

Timeboard with 5 Minute interval:

Graphical user interface, application

Description automatically generated

I was not able to see the Hourly Rollup Sum of my\_metric in the 5-minute, so I looked at the 1-hour timeframe.

Graphical user interface, application, website

Description automatically generated

Initially I couldn't find were to add the comment to notify myself. I first clicked the gear in the right-hand corner to see if there was a comment option. After about 5 minutes I found that by clicking the graph allows you to create a comment and send the notification. Below is the screenshot the graph using the @ notation to send to myself.

Graphical user interface

Description automatically generated

**Bonus Question**: **What is the Anomaly graph displaying?**

I couldn't find a postgresql metric that oscillated in such a way that the gray band on the metric would show up in the 5-minute interval. So I used the anomalies() function to graph the average of the postgresql.percent\_usage\_connections metric over time to include when I first integrated Postgres with my Agent. By using a 7-day timeframe I was able to see the Anomaly graph displaying the gray band around the metric showing the expected behavior of a series based on the past. Below is a screenshot of this:

Graphical user interface, application

Description automatically generated

## 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.**

Below is the exported JSON of the Metric Monitor:

"

{

"id": 25507064,

"name": "My Metric Threshold Levels",

"type": "metric alert",

"query": "avg(last\_5m):avg:my\_metric{host:vagrant} > 800",

"message": "{{#is\_alert}} my\_metric reached a critical level of {{value}} at IP Address: {{host.ip}}, which is unusually high. Check it out ASAP! @luke7@vt.edu {{/is\_alert}}\n{{#is\_warning}} my\_metric reached {{value}} at IP Address: {{host.ip}}. Keep an eye on it. @luke7@vt.edu {{/is\_warning}}\n{{#is\_no\_data}} No data coming from my\_metric in the past 10 minutes.\n1. Check if {{host.name}} is running\n2. Check my\_metric code in configuration file to see if anything has been changed\n @luke7@vt.edu \n{{/is\_no\_data}}",

"tags": [],

"options": {

"notify\_audit": true,

"locked": false,

"timeout\_h": 0,

"new\_host\_delay": 300,

"require\_full\_window": true,

"notify\_no\_data": true,

"renotify\_interval": "0",

"escalation\_message": "",

"no\_data\_timeframe": 10,

"include\_tags": true,

"thresholds": {

"critical": 800,

"warning": 500

}

}

}

"

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

Graphical user interface, text, application, email

Description automatically generated

**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,**
* **And one that silences it all day on Sat-Sun.**
* **Make sure that your email is notified when you schedule the downtime and take a screenshot of that notification.**

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

When creating the two scheduled downtimes, each the initial notifications showed the correct amount of time, but the start and end hours were shifted and I'm not sure why. Screenshots of each downtime setup can be seen below:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

## Collecting APM Data:

**Given the following Flask app (or any Python/Ruby/Go app of your choice) instrument this using Datadog’s APM solution:**

**I instrumented the app using Python and the ddtrace-run method. Below are the steps and code I used in the Vagrant Agent to collect APM data:**

First, I installed the following libraries:

"pip3 install flask"

"pip3 install ddtrace"

In the datadog.yaml file I uncomment the following:

"

apm\_config:

enabled: true

"

Next, I created a Python file for the app, called my-app.py, and inserted the following code into it:

"

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')

"

Finally, after restarting Vagrant, I ran the app:

"ddtrace-run python3 my-app.py"

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

A Service is a group of processes, such as queries, endpoints, or jobs. Whereas a Resource is a specific process such as a query, endpoint, or job that is instrumented by a service.

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

https://p.datadoghq.com/sb/95557gac4g6agrso-2020678a12480ea825d0308892b21506

Graphical user interface, application

Description automatically generated

## Final Question:

**Datadog has been used in a lot of creative ways in the past. We’ve written some blog posts about using Datadog to monitor the NYC Subway System, Pokemon Go, and even office restroom availability!**

**Is there anything creative you would use Datadog for?**

I think it would be really interesting if Datadog could monitor the wait time at airports, from the time you walk into the airport to the time you board. As opposed to showing up unnecessarily early, I think it would be efficient if Datadog could give a break down visualization of each step of the process - from checking in, waiting in line to get to security, getting through security, the time it takes to get from security to your gate, etc. Every airport is different in terms of capacity, the amount of foot traffic, speed of checking in, speed of security from the time you put your items on the belt to the time you get through and reclaim your items, distance to gates, etc. and I think a real-time monitoring application for the consumer would make the airport experience more efficient and less stressful.