Let's Try Fog Computing: Storing Fitbit Data in Redis

Jeff Stevens
Denison University Math and Computer Science
Granville Ohio, USA
steven_j1@denison.edu

Abstract -- One of the advantages of the concept of fog computing is the ability to reduce the amount of data that is sent to the cloud as to increase efficiency regarding data processing done in the cloud. This can be accomplished with the model system of an Internet of Things device that creates and sends data to a fog node, which then sends processes data and sends it to a cloud node. In this paper, I implement this model of the aforementioned fog computing system by taking a Fitbit wristwatch, aggregating its data in a fog node Redis database, and have it prepared to be sent to the cloud.

Keywords – Internet of Things, Redis, Docker, Fog Computing

Introduction

The world we live in today is densely populated with electronic devices and sensors and presents a phenomenon which has been come to be referred to by the term *big data.* Merriam Webster defines big data as "an accumulation of data that is too large and complex for processing by traditional database management tools". Considering that modern computers can reliably perform billions of operations per second, the concept of big data can imply a data set that approaches infinity with respect to the quantity of its contents, as our already powerful means of processing data cannot hope to confront a workload that large. In order benefit from the generation of this data, there must exist a way to reliably process and manage it.

Fog computing has been proposed as such a way to be able to make management of big data more feasible. Expanding on the analogy of the cloud, fog computing refers to the use of cloud computing methods, but vastly increasing the number of nodes and distributing them geographically close to devices for the benefit of low latency communication time between devices and the nodes. Essentially, we can make big data more manageable if we do processing work on the fog nodes before it is sent off to the cloud. The goal of this paper is to use this model to implement and demonstrate fog computing for individuals who are learning about the model themselves so they might also be able to actualize the model in a similar way described in this paper by following the designs and methods used here.

In this particular implementation of fog computing, a Fitbit blaze wristwatch was used as the Internet of Things device, the fog node was a Raspberry Pi 3, and Redis is used as a database software. Python is used to manipulate the Redis database. While this implementation does fulfill all the roles within the model of fog computing, the devices used do not qualify for a true fog system, as the Fitbit blaze wristwatch data was not accessed directly by the Raspberry Pi 3, but rather through the Fitbit web API. This means that the data is already located in Fitbit's cloud and then is pulled from the cloud to the Raspberry Pi 3 fog node. Therefore, benefits of reduced latency, ubiquitous service provided by a large distribution of fog nodes are not seen within this implementation.

If readers wish to reap all of the benefits that the concept of fog computing presents within their own implementations of this model, they must be able to access the data provided by their Internet of Things device directly and have their fog node(s) within a reasonable geographic locality of their device as well.

Design & Methodology

The Fitbit Blaze wristwatch collects a large variety of data, including heartbeat, sleep cycles, calories burned, steps taken, and time that the wearer has been active during the day. This device has not been designed to have its data directly interfaced with developers though, as Fitbit directs any interaction with the data of their devices through the Fitbit web API. This API can be used by navigating to the developer section of the Fitbit website and registering an application with the API. Each application produces a unique link that can be sent to users of Fitbit devices which allows them to authorize requests of their data, which is delivered in JSON file format. These requests are made using the well-known requests library in Python.

Redis is one of the most popular key-value databases that exists at the time of writing [2]. Due to its popularity and the scholarship of learning how to navigate a standard, conventional key-value database, I decided use Redis to act as a way to store the data queried by the Fitbit API and use it as the setting to aggregate the data for preparation of the data's transmission to the cloud. The Redis database ran within a Docker container and was interfaced with using the redis-py python library. The JSON data queried by the Fitbit API and was managed using the python json library and then placed into the database using this redis-py library. The Raspberry Pi 3 fog node hosting the API queries and the Redis database was running CentOS 7 as an operating system.

After having our data aggregated into the Redis database, the data is able to be transformed by means of deriving new data with averages, medians, and many other statistical methods. After processing, the fog node queries the cloud node to see what data should be sent to the cloud. In this paper, I will refer to the data preferred by the cloud *cloud data requests*. Cloud data requests will be able to be changed without interacting with the fog node at all, as the fog node will read these data requests from a webpage supplied by the cloud node which explicitly states what keys on the database should have their values sent. The data is then serialized into JSON format and is sent to the cloud with an HTTP POST request.

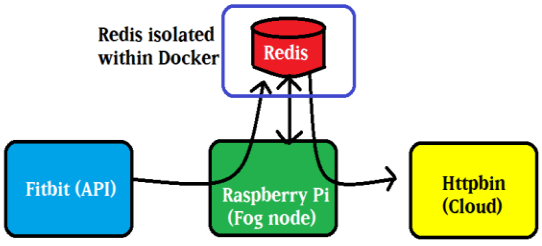

Figure 1: Data flow in this fog model

Implementation

In the process of implementing this model, as the case with all other people in efforts of software development, I encountered errors. The most noteworthy problem consisted of not being able to manipulate the Redis database within Docker from python. As I was sharing a machine with other students who also ran Redis databases of their own, I had to ensure that the Docker container that Redis operated inside of had a unique network port assigned to it which could be addressed by python commands for manipulating the data. This could be done when creating the container for the Redis database with the following command:

```
sudo docker run --name database -p\
xxxx:6379 redis
```

You can swap out a different name for "database" and you must replace the "xxxx" with a number between 0000 and 9999. This is the port number that is also supplied to python to make sure that any commands made in python are sent to the database as well. Note well that the number that you select for the port must not already be in use by another application. In Docker, users can view which ports are in use by all of their containers. If the reader is unsure of what ports or devices are being communicated to and from, tshark is a free software I recommend using for tracking all of the network traffic on one's device.

Other major stages of the implementation included the first reception of JSON data from the Fitbit API and the decision of what to do with the data once it was finally stored in Redis. The Fitbit API accepts queries by the use of the curl tool in Linux. In order to integrate this with the python requests library, I used the referenced website to convert my special curl command given by the API to a requests.get() operation [3]. 44 variables were obtained and stored in Redis from my device on a single API query, so I decided to only work with a few of them on the principle that fog computing reduces the amount of data sent to the cloud. I derived a new data I called "averageDistanceWalked" by multiplying the data for "strideLengthWalking" by the data for "averageDailySteps". The result was a new piece of derived data, "averageDistanceWalked" in inches. This is an example of data processing that could have been done in the cloud which is now completed in a fog node. Having done this, I will reduce the overall work needed to be done in the cloud. In other implementations of this model, derived data should be created in a fog node if possible.

After this data is stored and processed, it is ready to be sent to the cloud. The cloud data requests were hosted on a GitHub webpage (see figure 3). The

Figure 2: API query and terminal output from running the data processing and data transmission program on the fog node

fog node sent a GET request to this webpage and obtained the keys in the Redis database which matched the information received from the cloud data requests page using the beautifulsoup python library. In this way, theoretically, to get different types of data from the fog nodes, only the cloud data requests webpage needs to be updated, rather than all of the fog nodes. The focus of this project was not necessarily focused upon managing data in the cloud, so I did not make a cloud server database and ended up sending the data in JSON format with a HTTP POST request to a website called httpbin. Httpbin is a website that echoes the data sent to it in POST requests in its responses, so it is a great tool to test the transmission over data over the internet and works well for the purposes of this project. Should a cloud database be desired to be implemented in the reader's fog computing system, a solution I would recommend is to run an apache server on the Google Cloud platform which could handle the data transmission [4].

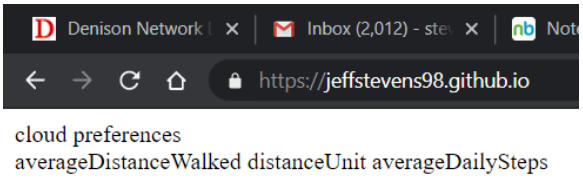

Figure 3: Cloud preferences webpage

Results

The outcome of this project was a successful implementation of a not completely true fog computing model, due to the data accessibility complications with the Fitbit device. The data was moved from each discrete storage device to the next successfully and resulted in a reduction of the amount of overall data that needed to be processed after it was sent to the cloud. The methods learned here in this very simple version of a fog computing and data aggregation system can be applied for industry level use and scale to accommodate more than one Internet of Things device and include more fog nodes for any purpose desired.

As no specific data analytic insights are produced by this paper, it should be clear to the expert novices that this paper is intended for that the result of this paper extends to what developers create using this paper as a guide. To further demystify the process of what was done in this paper, the python program that accomplishes the arc of this project's functionality is publicly available on my Github page at the link below:
https://github.com/jeffstevens98/fogProject

Related Works

For readers interested in fog computing who want to study the subject further at an entry level with a level of competency, having understood how

the model can be implemented, I have included a few papers that may be productive directions to take after reading this paper.

One direction to take after seeing a simple implementation of a fog computing system is to allow oneself to take a step back and view the general work done in the field, with special consideration given to the Internet of Things devices used in combination with fog computing. “Fog Computing and Its Role in the Internet of Things”[5] is a paper that delivers that exact experience. Experts declare that the solutions provided by the cloud are not always satisfactory in latency sensitive contexts and then explain how fog computing delivers better solutions in many use cases. It is a short read, at only three pages, and includes many interesting, higher-level papers as references which one can follow from there to keep exploring the subject at a high-academic level.

If theoretical discussion is something you would like to avoid, no problem. A specific applied project to dive into is the creation of smart grids; a modern industrial electricity distribution design which utilizes the power of the cloud to allow power companies to monitor production, consumption, and pricing of electricity in real time. The paper “A Fog Computing Based Smart Grid Model”[6] makes use of the concepts of fog computing to improve upon the system, by breaking down big data supplied by power grids, increasing privacy of company client power usage data, decreasing latency of data transfer, and allowing for location specific services due to the liberal geographic distribution of fog nodes.

Lastly, if one wants to explore an advanced application combining fog computing and computer vision, look within “Dynamic Urban Surveillance Video Stream Processing Using Fog Computing”[7]. The authors present a system architecture model which should be very familiar to readers of this paper. The job of the fog nodes in this paper, after receiving the data from the surveillance cameras, are to process the video before it is sent to the cloud where it can be viewed by a user. Just like we have demonstrated here in this paper, fog nodes can be especially useful in taking on workload that would otherwise be passed down the line to the cloud. The application for this paper outside of fog computing is a fascinating, yet hard to comprehend from a beginner’s perspective, use of computer vision algorithms to track objects in the gathered surveillance data.

Conclusion

Fog computing has been demonstrated to be a simple solution of great interest when it comes to finding ways deal with the phenomena of big data. Who knew that if we just brought the internet physically closer together and had it everywhere, it would actually work better? Getting the data to flow from sensor, to fog node, to the cloud is just the beginning of the long and ever-increasing list of possibilities of projects that employ fog computing, as one can see here from a couple of the wild ideas that have begun to make reality resemble a science fiction movie. Taking the lessons learned from getting started with fog computing in this project, what will you make?

References

1. “Big Data.” Merriam-Webster. Merriam-Webster. Accessed May 1, 2020. https://www.merriam-webster.com/dictionary/big%20data.

2. “DB-Engines Ranking of Key-Value Stores.” DB-Engines, May 2020. https://db-engines.com/en/ranking/key-value+store.

3. “Running a Basic Apache Web Server | Compute Engine Documentation.” Google. Google, n.d. https://cloud.google.com/compute/docs/tutorials/basic-webserver-apache.

4. Carneiro, Nick. “Convert CURL Command Syntax to Python Requests, Ansible URI, MATLAB, Node.js, R, PHP, Strest, Go, Dart, JSON, Elixir, and Rust Code.” Convert cURL command syntax to Python requests, Ansible URI, MATLAB, Node.js, R, PHP, Strest, Go, Dart, JSON, Elixir, and Rust code. Accessed May 8, 2020. https://curl.trillworks.com/.

5. Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. “Fog Computing and Its Role in the Internet of Things,” n.d.

6. Okay, Feyza Yildirim, and Suat Ozdemir. “A Fog Computing Based Smart Grid Model.” 2016 International Symposium on Networks, Computers and Communications (ISNCC), 2016. https://doi.org/10.1109/isncc.2016.7746062.

7. Chen, Ning, Yu Chen, Yang You, Haibin Ling, Pengpeng Liang, and Roger Zimmermann. “Dynamic Urban Surveillance Video Stream Processing Using Fog Computing.” *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, 2016. https://doi.org/10.1109/bigmm.2016.53.