SYSTEM DATA AGGREGATION FOR LINUX-BASED APPLICATIONS

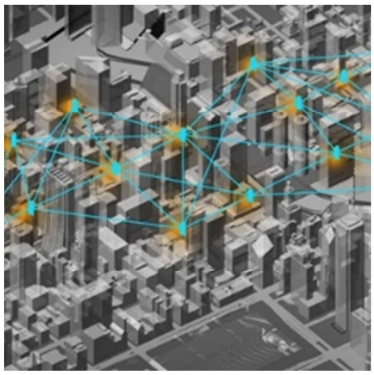
or: Waggle Node Metrics API

Adam Morrison

For the past 10 weeks I have worked closely with the Waggle Team, a part of the Array of Things Project to research System Data Aggregation for Linux-Based Applications culminating in the creation of an API.

INTRODUCTION

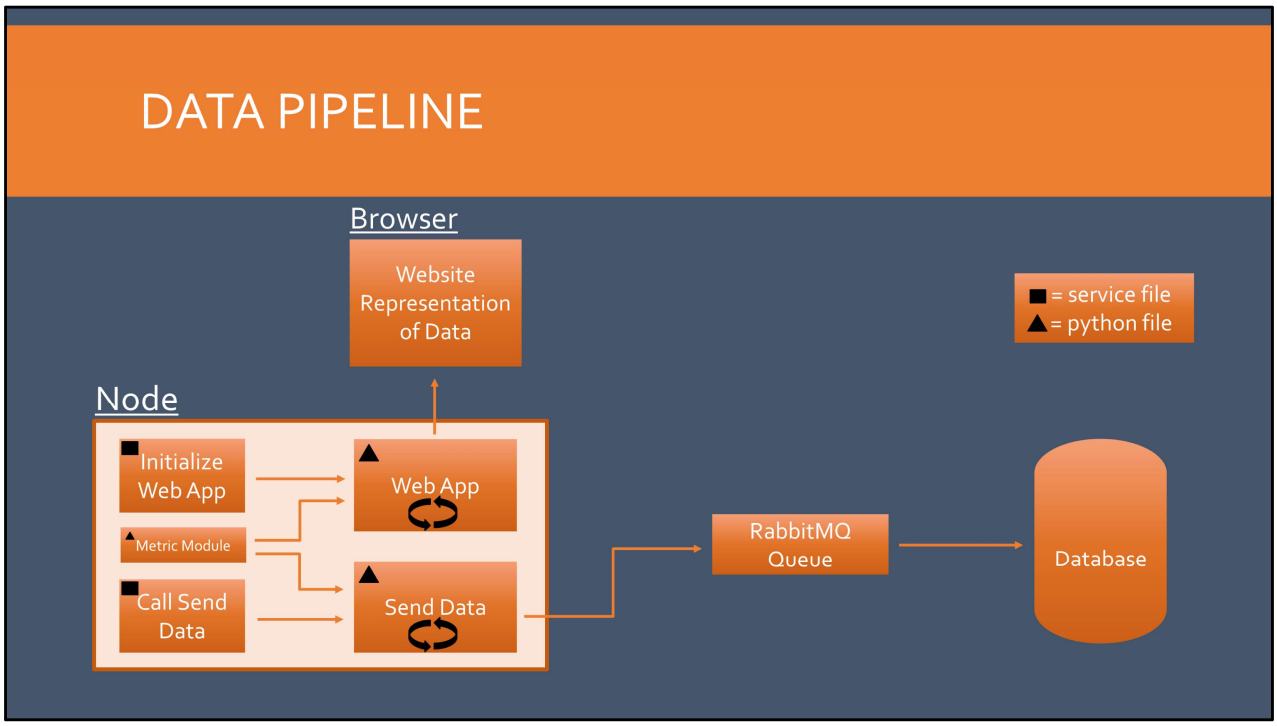
- Need for more accurate information about cities
- Waggle Project – Array of Things
- Nodes – collect city data
- Single-board computers running Linux



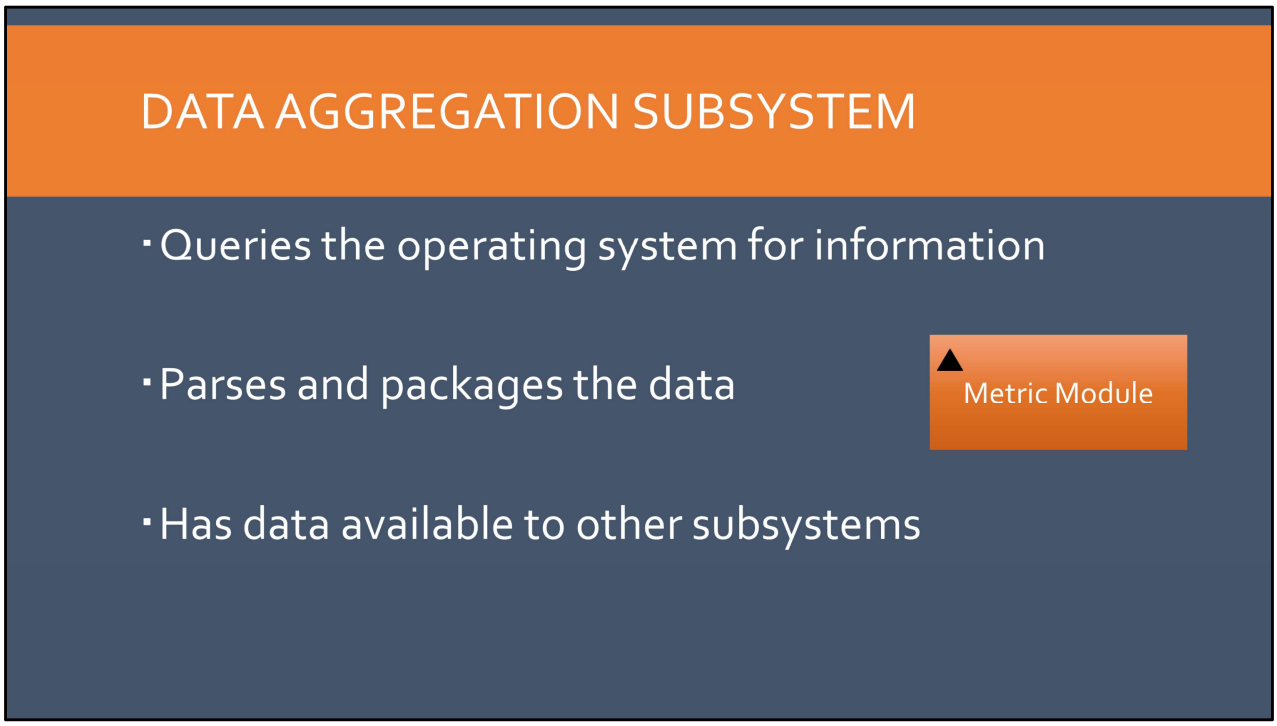
The Array of Things project developed at Argonne National Laboratory was started to accomplish the task of urban sensing paired with data collection and analysis to obtain more accurate information concerning cities and certain wildlife areas. The project consists of "…a network of interactive, modular sensor boxes" that are spread throughout a city, such as Chicago. These sensors, also called Nodes are placed on light poles and around a city, and are able to measure, record, and transmit data relating to the surrounding environment. The "brain" of the Nodes consist of two embedded single-board computers. These computers use the Linux Operating System, a free and open source software that allows developers to control and change core functionality of the machine.

However, despite the extensive software functionality of the Waggle Nodes, there was still a need for a platform to collect data related to the computer hardware and software itself. Although there are tools in place to find system metrics to diagnose problems, no comprehensive monitoring and recording of these metrics existed, and there was no easily accessible and readable aggregate list of these metrics. In the event of a Node failure, a singular developer would be required to investigate and diagnose a Node individually which requires time and resources. Using Linux command line tools and knowledge of computer systems, one developer could take a long time to diagnose a problem, in addition to the time needed to correct the issue. The research project, therefore, focused on the creation and implementation of a system software monitor checking important metrics relating to a Node at the hardware and software level and the aggregation of this Linux system data.

The proposed solution to the problem was to create an Application Programming Interface (API) consisting of three subsystems that could accomplish three tasks. These tasks were collection of diagnostic system data from a Linux based single-board computer, transmittance of this data to a remote database for storage and analysis, and presentation of the information to a human in an easily readable and accessible web page format. Advantages of this system would be that the list of metrics related to the single-board computer would be in a central location, allowing information to become easily accessible and readable. In addition, due to the web page format of the information, users with little experience in the field of computers could diagnose issues, thus reducing time and resources needed to find and fix problems. In a larger sense, the simplification of reduction of problems in Nodes opens the Array of Things project to focus on other areas of improvement, as well as continuing to serve the cities with important data. Thus, the proposed API solution would be a step in correcting issues with Nodes at a hardware and software level, but also issues within cities that can be solved by implementing these Nodes.

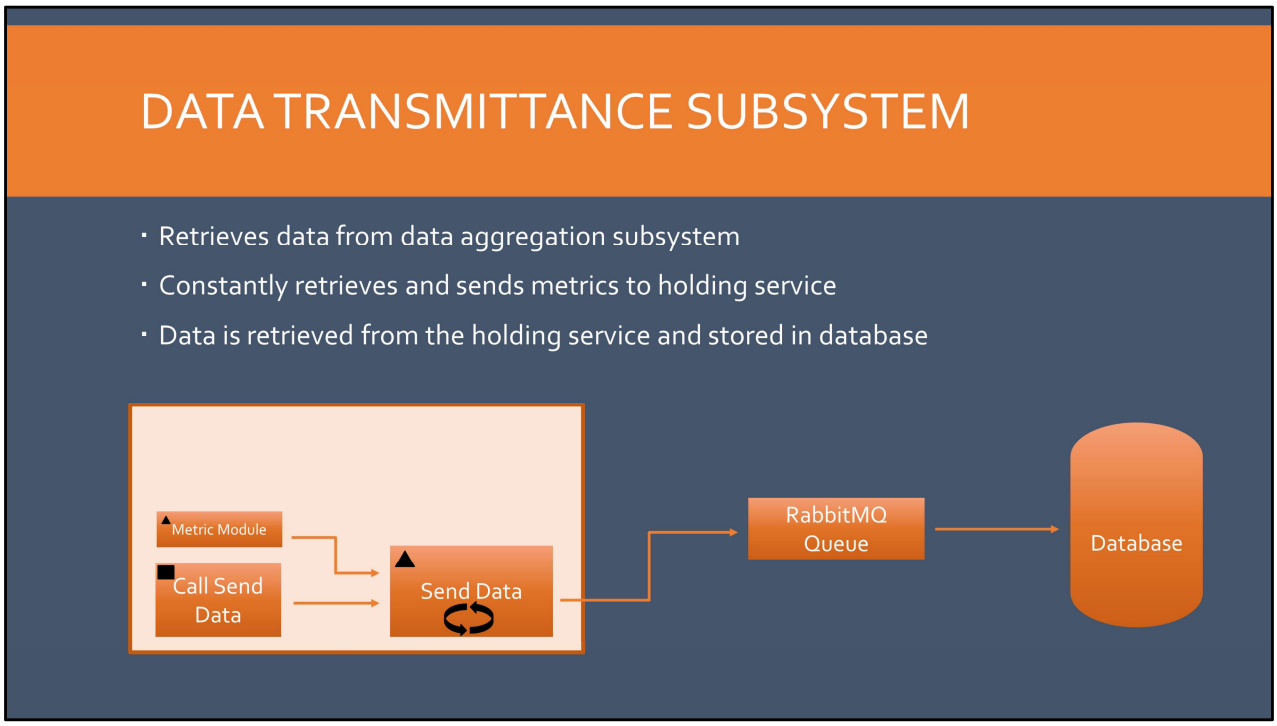


The flow of data in this API would start with the data aggregation subsystem in the Metric Module box which would collect the computer metrics. The data transmittance and web app subsystems, represented by the Send Data and Web App boxes, respectively, would be started by Linux system services on computer boot up. After boot up, each subsystem could then query the data aggregation subsystem for the computer metrics and use them according to their functionality. The data transmittance subsystem (Send Data box) would send the information to be stored in a remote database, and the web application (Web App box) would show the information on a web page for the local user.



The first subsystem of the API is the data aggregation subsystem, which would query the operating system for various metrics regarding the single-board computer. Some of these metrics included: memory information, disk (hard drive) usage and information, processor information, uptime of the computer, connected USB devices, running services, and uptime of services.

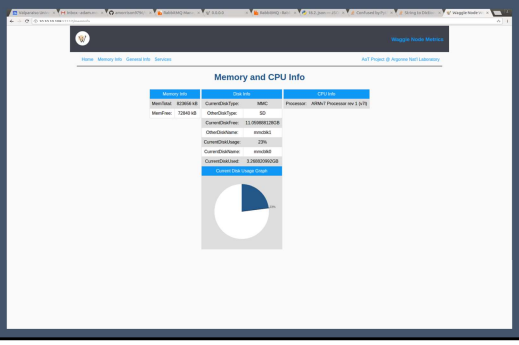
This subsystem contained a single Python file represented by the Metric Module Box. This file contained functions that held the capability to query the computer for the aforementioned metrics. In order to retrieve metrics, the Python file would employ two existing modules called subprocess and re (regular expression). The subprocess module could call Linux commands from the Python code, which simulated a user entering a command in to the Linux command line. In addition, the re module helped to parse the data that was retrieved using the subprocess module. Using regular expressions on the retrieved data strings allowed patterns in the data to be found for processing and detection of specific, vital parts of the retrieved information. The information was then packaged into a Python dictionary and formatted as JSON for easy use by other subsystems.

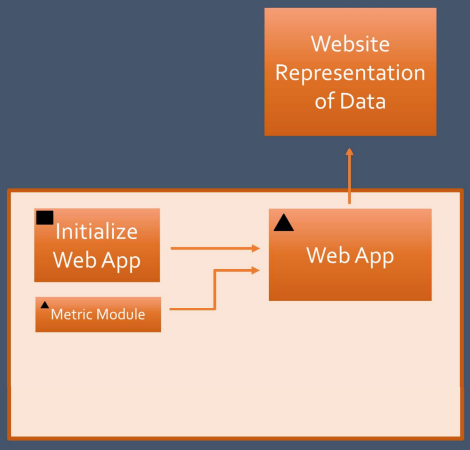


Since the data aggregation subsystem could be queried for information at any moment in time, it would only collect data when it was called. The Python file sendData.py, as represented by the Send Data box would be able to retrieve a Python dictionary of the computer metrics in JSON format from the data aggregation subsystem. The data transmittance subsystem's pipeline begins when the service shown by the Call Send Data box starts the Send Data Box Python file at computer boot up. This Python file would then run continuously, communicating and sending the retrieved metrics over the network at a user specified time interval to a holding service called RabbitMQ. This service would then hold the data until software written by another project member would retrieve the information, delete it from the queue, and store it in the remote database.

WEB APP SUBSYSTEM

- Retrieves data from data aggregation subsystem
- Creates web server on a local network
- Displays information on web pages to user





The subsystem that encompassed the web view of the aggregate data begins when the service represented the Initialize Web App box would enable the web application on computer boot up. The web application represented by the Web App box was written in the Python programming language, using a micro-framework for webpages called Flask. When started, the Python file would begin a server on the localhost network, i.e., itself, and would then serve web pages to a user connected to the network. HTML and CSS files were called from within the Python file to render the web pages. Using a web browser on the local network, a user could view the metrics gathered by the data aggregation subsystem on the served web pages.

CONCLUSION

- The API was created and worked as expected
- Three subsystems managed their respective functionalities
- Implications are both large-scale and small-scale

At the end of the 10-week research project period, the proposed API solution of a system software monitor retrieving, sending, and displaying important computer metrics related to a Node was successfully accomplished. The implemented code of the three subsystems of the API were able to run on a Node, and each succeeded in completing the task that it had been designed to manage. The implications of this research project are both small-scale and large-scale. The constructed API simplifies the diagnostic process for developers and enables problems to be fixed earlier in the development process. In addition, it creates an aggregated location for the metrics it gathers to be viewed. Due to the addition of the API to Waggle Nodes, the over-arching Array of Things project goal of measuring and recording city data can be reached much quicker. The development of this API will enhance the capabilities of Node developers, propelling the project forward, thus allowing the information gathered by Nodes to have a positive impact on the served cities at a faster pace.