Assignment 1

1. Summary

The driver program presented herein is meticulously designed to interface with Sysdig. The primary objective of this program is to periodically extract specific metrics related to workloads, visualize this data for intuitive comprehension, and archive the raw metrics for potential future analysis. This report provides a walkthrough of the program's architecture, its functionalities and its significance for monitoring. The application that is being monitored is Acme Air that has been implemented on the OpenShift cluster. The deployment of the microservices has been modified to expose the required jmxremote service for better metrics collection. The exposed metrics can be viewed on the *IBM cloud monitor*.

2. Program Architecture & Design

The program's foundation is built upon a suite of essential libraries, including **IbmAuthHelper**, **SdMonitorClient**, and **matplotlib.pyplot**. These libraries ensure that the program operates seamlessly, to provide all the necessary tools that are needed. Upon initialization, the program sets up configuration parameters such as URL, APIKEY, and GUID. These parameters are instrumental in establishing a secure connection with Sysdig and an instance of SdMonitorClient is then created.

A comprehensive array named **metrics** serves as the program's backbone, detailing the metric IDs and their associated aggregations. This array acts as a roadmap, guiding the data extraction process to fetch only the most pertinent information. The major metrics that are included are of the JMX and Sysdig CPU and Memory monitors. Now since the application consists of several microservices, these microservices are defined in order to be accessed and a nested dictionary, **data_storage**, has been architected to systematically archive the metrics data.

The function **fetch_data_for_workload** is the main driver function that collects the relevant metrics as required. The function runs for 10 minutes (600 seconds) and collects data at an interval of every 3 seconds. The duration of interval is kept in accordance with the interval period of the IBM cloud monitor. The configuration of the cluster name, namespace and the microservice is mentioned as the filter while getting the data. As there are different microservices running at the same time, we have used **threads** in order to collect the data simultaneously from each of them. As the metrics are collected, they are all appended to the data_storage for analyzing later on.

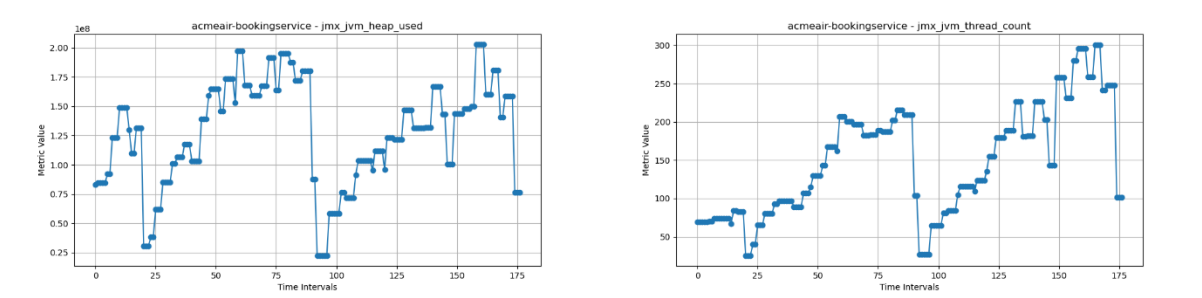
In order to get the metrics, we have setup JMeter to simulate user traffic. The initial configuration of the JMeter has been done accordingly. The number of consecutive threads were varied to get different load on the website. A maximum of 1000 threads were taken and considered as full load. The flight database was loaded with 500,000 samples randomly generated (*Generate data*). Once the test was started, it was monitored to ensure the queries that are being made take place successfully in order to mimic the network traffic.

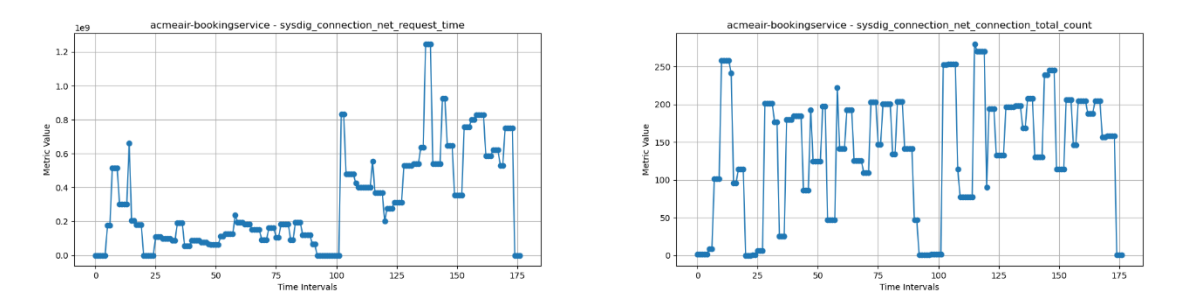
2. Visualization and Data Representation

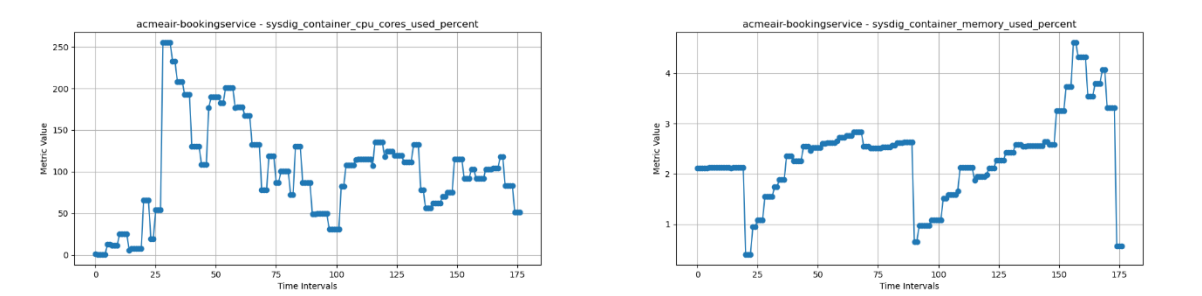
Once the data collection is done for each of the microservice and metric, the data is plotted with **matplotlib** giving it relevant service name and metricID. These plots are then saved as PNG images for visualization. In addition to this, the raw data that is collected is stored in the form of **json** file named collected_data.json. Further, an editing

file **jsonToCSV.py** is provided to convert the saved json data to CSV format. The output of this file has also been included as the report.

Some of the visual outputs have been presented below.







From *'net_connection_total'*, we can see that initially the program did not receive any requests. However, it subsequently received a high number of requests continuously until the end of the test. *'thread_count'* and *'heap_used'* also corroborate this observation. As the number of requests increased, the program needed to activate more threads to respond to the requests. During this process, the program had to allocate more heap memory. Therefore, the changes in *'net_connection_total'*, *'thread_count'*, and *'heap_used'* are roughly synchronized.

Next, looking at the *'net_request_time'* that reflects the maximum response time of the requests, we notice that it is significantly influenced by CPU utilization and memory usage. We can observe that CPU usage shows a gradual declining trend, while memory remains mostly constant, only showing an increase towards the end. The usage of CPU and memory is related to the number of requests. Looking at the response time trend, it aligns more closely with the trend of memory usage, indicating that the response time is more correlated with memory usage than with CPU utilization.

Thank you. *Group 11*

All the plots and raw data along with the scripts used have been provided separately. The codebase used has been updated to [GitHub repository] and is made up to date. In case of any discrepancy, kindly reach out at n2chhabr@uwaterloo.ca.