**UNIVERSITY OF WATERLOO**

Faculty of Mathematics

**[Insert Title Here]**

Bank of America Merrill Lynch

Toronto, ON

Prepared by

Kushan Zaveri

1B Computer Science

ID 20721646

Month Day, Year

**\*\* Letter of Submittal Not Included**

**Table of Contents**

**List of Figures**

**Executive Summary**

* the purpose of the report
* the key points of the analysis
* the highlights of the conclusions
* the highlights of the recommendations

1. **INTRODUCTION**

At Bank of America Merrill Lynch, the S1 team is developing an application that is used to book trades and create swaps, two types of financial transactions that the company is responsible for conducting for their customers. There are nine developers at the Toronto Global Business and Marketing Technology (GBAM) office who work on the S1 application. The project manager of the Toronto GBAM team, Michel Chicoine, is responsible for conducting team activities and ensures that they produce software at the highest quality. A current issue is that it is difficult to find sources of error and track the quality of the team’s output. There is a need to assess metrics relating to the performance of the S1 application in order to diagnose software issues on time and react to them promptly.

A member of the team is required to develop a monitoring application programming interface (API) that allows developers to publish data relevant to tracking the performance of the application. As described by “Improving API Usability,” an API allows developers to reuse code for their own purposes (Myers and Stolysddisoid). In this case, the monitoring API would make it possible for other developers in the team to create metrics without having to write new code in the future. This monitoring API would ultimately publish this data in an accessible and clear manner to provide the S1 team a real-time view of all system processes.

This report outlines the importance of tracking metrics and potential beneficiaries of this information. It will explain the technical process of developing the monitoring tool and the benefits it will provide team members.

1. **ANALYSIS**

In the Information Technology field, it is important that many aspects of software development are monitored. This may range from the productivity of the team to the quality and performance of the end product. The specific data that is monitored will vary depending on the requirements outlined by the project manager.

Due to the high failure rates in IT projects, specifically those dealing with software development, it is vital for the team to track key metrics (Marsanu, 2010). As observed at Merrill Lynch in the S1 application team and various other software development workplaces, bugs and errors come up quite regularly. By measuring certain attributes, it becomes significantly easier to track processes and thus identify and resolve problems in a timely manner.

* 1. **How the Monitoring API was developed:**

Whenever a task is performed in the S1 application, the software will output a message containing information relating to that task. The message will be in a format called “JSON”, a standard format used to transport data. A JSON message takes the following structure (Fig. 1)

Fig. 1 A sample JSON message from the S1 application.



**Value**

**Attribute**

As shown in (Fig. 1), a JSON message has attribute-value pairs. For example, “environment” is the attribute and “QA” is the corresponding value. This specific message would be sent by the application to indicate how much free memory (System.Memory.freeMemory) the application has in the “AMRS” region at a given time. The “value” indicates the amount of free memory available.

Since the S1 application already produces these JSON messages, the monitoring tool will need to adhere to the existing format and use this data. All JSON messages from S1 can be found on a messaging client called AMPS, a third-party software used to send and receive data over a server [CITATION]. The messaging client is organized in to channels, which are different locations that JSON messages come through. For instance, system events come through the “SYSTEM/EVENTS/JSON” channel (Fig. 2).

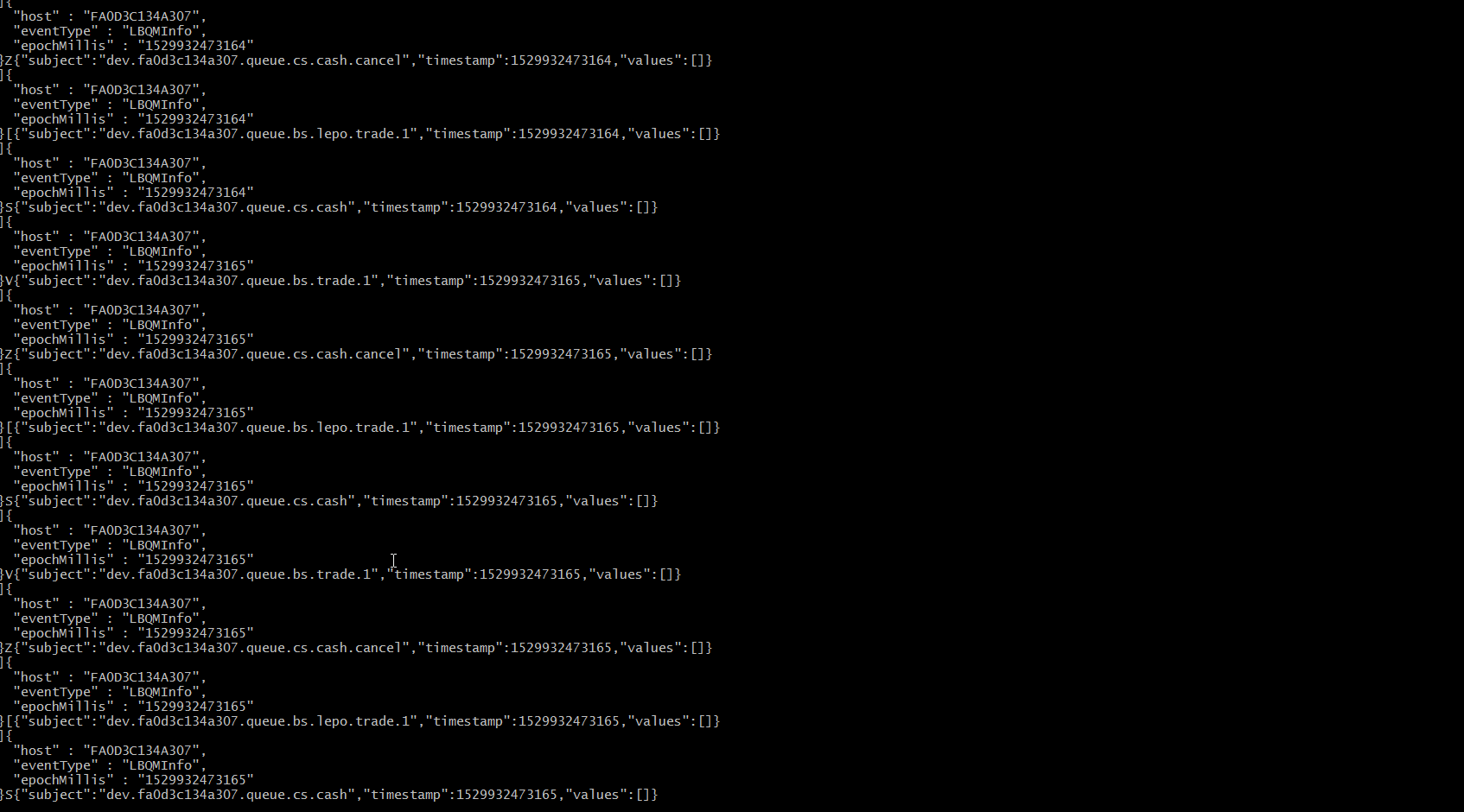


Fig. 2 A stream of JSON messages coming through the “SYSTEM/EVENTS/JSON” AMPS channel.

Once the raw JSON messages are on an AMPS channel, the data needs to be processed to create useful metrics. In order to handle the input, processing, and output of these metrics, a processing unit (PU) model is used. This model was created by the developers of the S1 team for the application itself and so it would be logical for the monitoring tool to build off this structure. The PU structure is as seen in (Fig. 3).

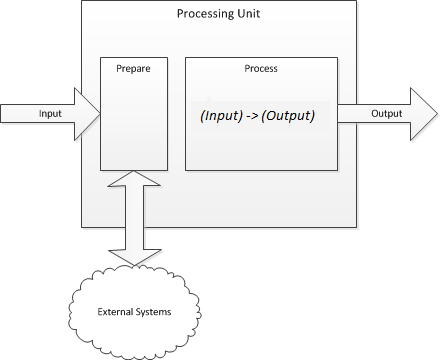


Fig. 3 The way in which a processing unit is structured.

At its core, a PU accepts data in the JSON format, prepares it for use, performs the processing required to obtain useful information, and will finally output the newly created data.

There are two main benefits to using this processing unit structure. First of all, the PU was designed to be daisy-chained, meaning that one PU can be linked to another. This approach allows the monitoring tool to break down the processing of one task in to many smaller tasks. The other advantage is that a PU conceals the complicated processing from the user. Thus, any member of the S1 team can provide the PU with an appropriate input and not have to worry about how the input is handled internally. This is a key feature when creating an easy-to-use API.

**Creating the Processing Unit:**

**Input:**

As mentioned above, all JSON messages are sent to an AMPS channel. The processing unit would need to “listen” to the appropriate AMPS channel to receive the JSON message. For every processing unit, there is a thread running on the server, waiting for new messages to come in on the AMPS channel.

**Preparing data:**

Before the code can interact with the received JSON message, it must be “de-serialized”. As described in “Beginning JSON”, De-serializing is the process of taking a JSON message from a certain point in time and converting it in to a data structure (CITATION NEEDED). Since Java is the primary language used by the S1 team, a Java de-serializer was needed. Hence, a tool called GSON was used to convert JSON to Java data structures. GSON provides many Java functions that allow users to easily parse a JSON file and place it in to a Java object. The following image compares what the corresponding Java object would look like for a given JSON object (Fig. 4). Note that all the attribute names in the JSON object match the variable names in the Java object.

Fig. 4 The corresponding Java object (left) for a given JSON object (right).

JSON OBJECT:

{

“region”:"AMRS",  
“environment”:"QA",  
“source”:"Payment Service",  
“name”:"System.GC.ps\_marksweep.time",  
“timeStamp”:1529946357411,  
“duration”:397

}

JAVA OBJECT:

public class DurationEvent {

public final String region;  
 public final String environment;  
 public final String source;  
 public final String name;  
 public final Long timeStamp;  
 public final Long duration;

}

Once the Java object is created with GSON, Java code can be used to perform the processing.

**Processing data:**

Once the JSON message has been deserialized, all the processing can be done on the newly created Java object. As shown in (Fig. 3) the processing stage takes in an “Input” and will produce an “Output”. The “Input” is the Java object created in the preparation step. In the processing stage, code is written to enrich the Input and create a new Java object. For instance, given a JSON object, the processing enriches it’s respective Java object and creates a new Java object (Fig. 5). This final Java object is the “Output”.

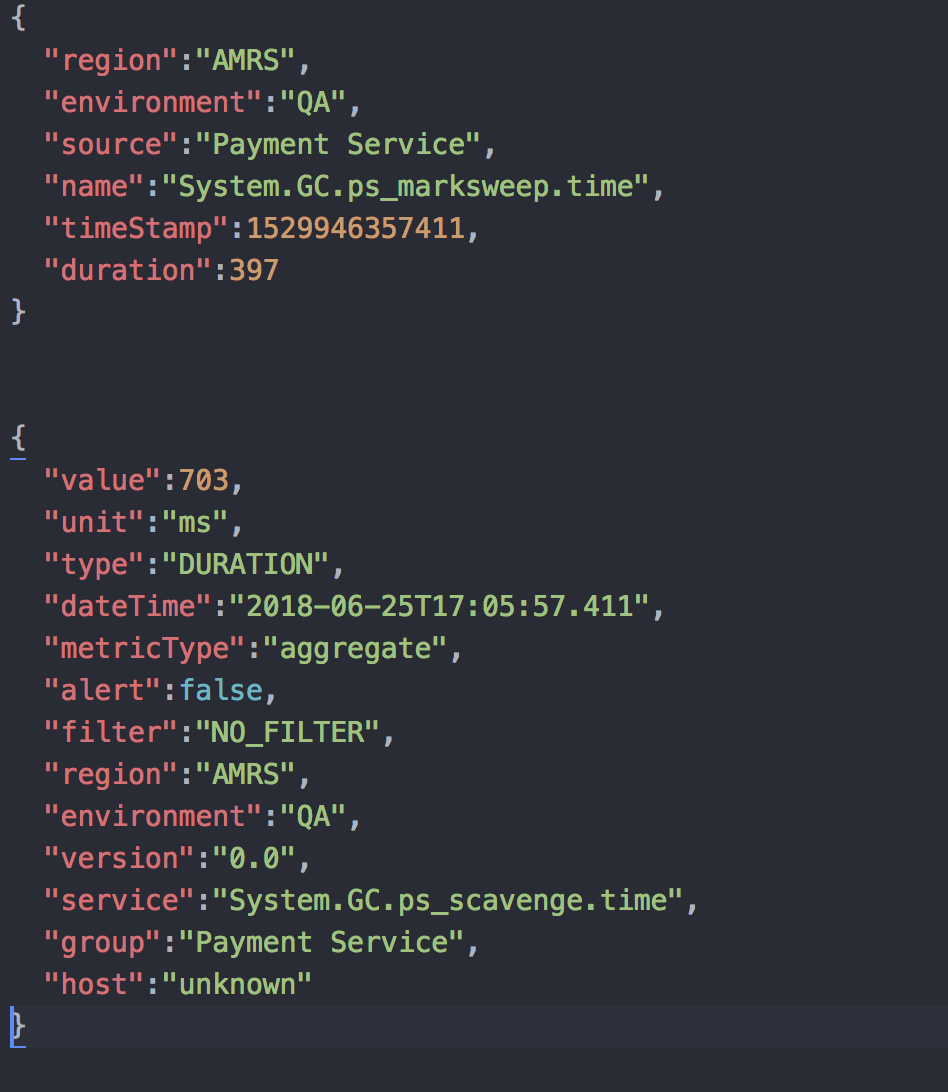


Fig. 5 The Input JSON object (left) is enriched with more attributes and values to give an Output (right).

**Outputting data:**

Once a Java object is created, there are many options for where it can be sent. If the Java object is being passed to another PU, it is converted to a JSON object through serialization, the opposite of de-serialization. In the final step of the monitoring tool, the Java object will typically be published to a database.

There were many options for which database should be used to store all the outputted data. Since all the data produced by the S1 application has a timestamp and because the monitoring API’s end goal was to provide a real-time view, a time-series database would be the ideal choice.

A time-series database is one that organizes data points by the time they occurred. InfluxDB was chosen as the time-series database for this project as it provides support for the Java programming language, meaning that Java objects can be published directly to the database. Thus, it would fit in to the PU structure that was already adopted. Furthermore, InfluxDB support is provided by Grafana, a graphing tool which will be described later. The data sent to InfluxDB is stored in a table (Fig. 6).

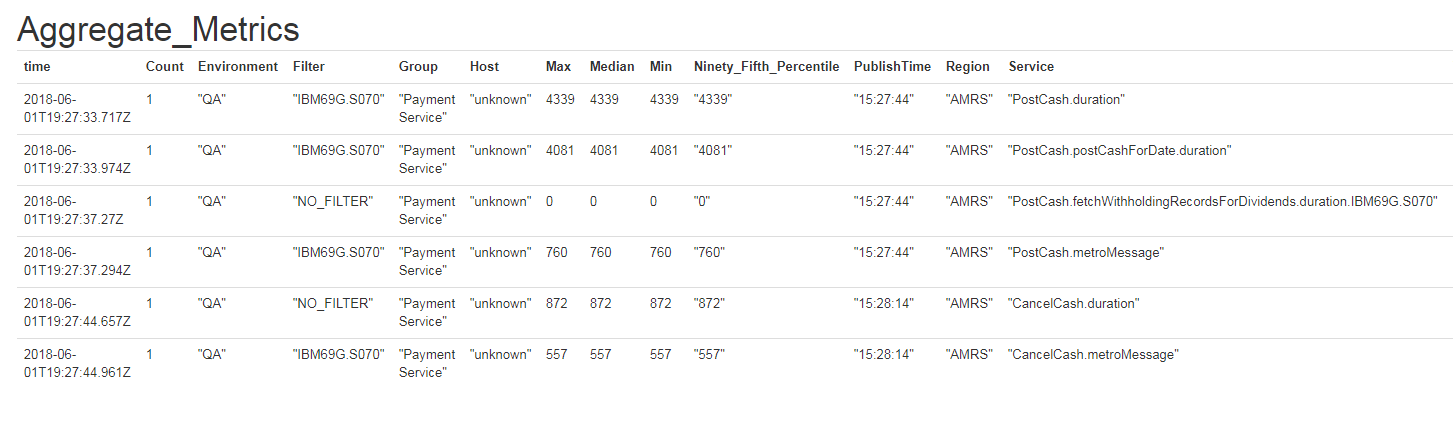


Fig. 6 A table of data that is published to the InfluxDB database from the monitoring tool.

The current model of the PU would write data directly to InfluxDB as soon as a JSON message was received. So, if the PU had to write 200 JSON messages to the database, it would have to connect to the database 200 times. This was found to be extremely inefficient, and it needed to be improved.

The solution was to create two processing units. The first PU would receive messages from the S1 application and only perform the processing on the raw data. This PU would not be responsible for outputting to the database. Instead, it would send a JSON message to another AMPS channel. The second PU would listen to this AMPS channel every thirty seconds and would solely be responsible for writing to the database. This way, only one connection is needed to InfluxDB every thirty seconds, which significantly improves the run time.

**Visualizing data:**

The previous section describes how data is monitored, processed, and stored, however it needs to be made accessible to the project manager and team members. In order to present the data, a tool called Grafana is used to generate graphs and displays to provide an overview of the information in the database. As mentioned previously, Grafana has support for InfluxDB, meaning that data from the InfluxDB database can directly be accessed by Grafana, with no external setup (CITATION?). Furthermore, since InfluxDB is a time-series database, each data point has a corresponding time. This allows Grafana uses time as the independent variable for all of the graphs. In order to access specific data from InfluxDB, SQL, a standard language used to query data, is used. The following query would be used to graph the “Median” value from an “Aggregate\_Metrics” table in InfluxDB (Fig. 7).



Fig. 7 A sample SQL query to graph the Median field from the Aggregate\_Metrics table in InfluxDB

In order to organize the metrics and provide multiple views for different users, Grafana provides the ability to create dashboards (see Appendix A for a sample Grafana dashboard), which contain graphs related to a certain topic. For instance, one dashboard might contain information related to memory usage and another containing processing times for various processes. Another feature in Grafana is the ability to filter and narrow down data by a certain time/region/environment/etc. This provides added functionality to the users of the data.

The end result is a set of dashboards that provide developers and the project manager with a general overview of various metrics from the S1 application.

**Disadvantages:**

The current system is very versatile and provides the users with a good solution to their problem at hand. However, there are some disadvantages with the current system.

First of all, InfluxDB has a limit to how much data can be stored. Due to the large operation of Bank of America Merrill Lynch, all data in the database needs to be deleted after approximately a week to prevent InfluxDB from reaching full capacity. There are a few ways to circumvent this issue, by converting the raw data in to less-precise data after a couple days. For instance, rather than keeping 10 data points in the database, it can be replaced with 1 data point containing the average of all 10. The other issue is that Grafana graphs and dashboards must be created manually. The monitoring API only allows for users to easily write data to the database. The user is completely responsible for the design and writing queries to extract information and present it in an accessible manner. In the future, there will be some hassle when it comes to logging more metrics in Grafana.

**2.2 – How the New System Affects the Workplace**

There are many people that can benefit from tracking and monitoring data related to the S1 application. Internally, the project manager can use software metrics to find the root of an issue promptly after it occurs by investigating the raw data logged by the monitoring API. This information can then assist them in assigning tasks or providing stakeholders with accurate explanations of why a failure might have occurred. Finally, it provides an objective way of tracking the team’s progress and performance, which is useful to both the manager and the developers (Jethani, 2012).

Apart from the management of the team, the developers in the S1 team have the responsibility of creating software of the highest quality possible. By exposing statistics that relate to application performance, developers will be able to quickly assess the overall status of the S1 software to see that it is performing well. Furthermore, the system will provide an accessible way to see the impact of a developer’s changes on the overall system. For instance, if a developer were to modify the application’s process of opening a new transaction, they could see if it has an effect on the process run times (how long it takes to execute). In turn, the S1 team will be able to make smarter decisions when it comes to developing software.

As a result of the improved management and operation of the development team, the users of the application will notice enhancements in performance and fewer bugs. It is also in Merrill Lynch’s best interest to improve the quality of their products and the productivity of their employees, and measuring key attributes are one way to reach this goal [provide a citation].

1. **CONCLUSIONS**

The conclusions that can be made from the analysis of this problem are:

1. A key component in developing an API is making sure that it is versatile while also being easy-to-use and interact with for developers. The monitoring API accomplishes this by providing simple PUs that can be used by any developer on the S1 team.
2. The technology used by the monitoring API such as InfluxDB and Grafana have certain drawbacks that might make it difficult to visualize data in the future.
3. There is great use in logging metrics and exposing data for a team as it provides key insight to company processes and can lead to improved product quality and team productivity.

References

<https://link-springer-com.proxy.lib.uwaterloo.ca/article/10.1007/s13198-012-0101-1?institute=OCUL.waterloo>

Pocatilu, P. (2007). IT Project Management Metrics. *Revisita Informatica*

*Economica*, 4 (44), 122-123.

Marsanu, R. (2010). Project Management Metrics. *Oeconomics of Knowledge*, 2 (2),

10.

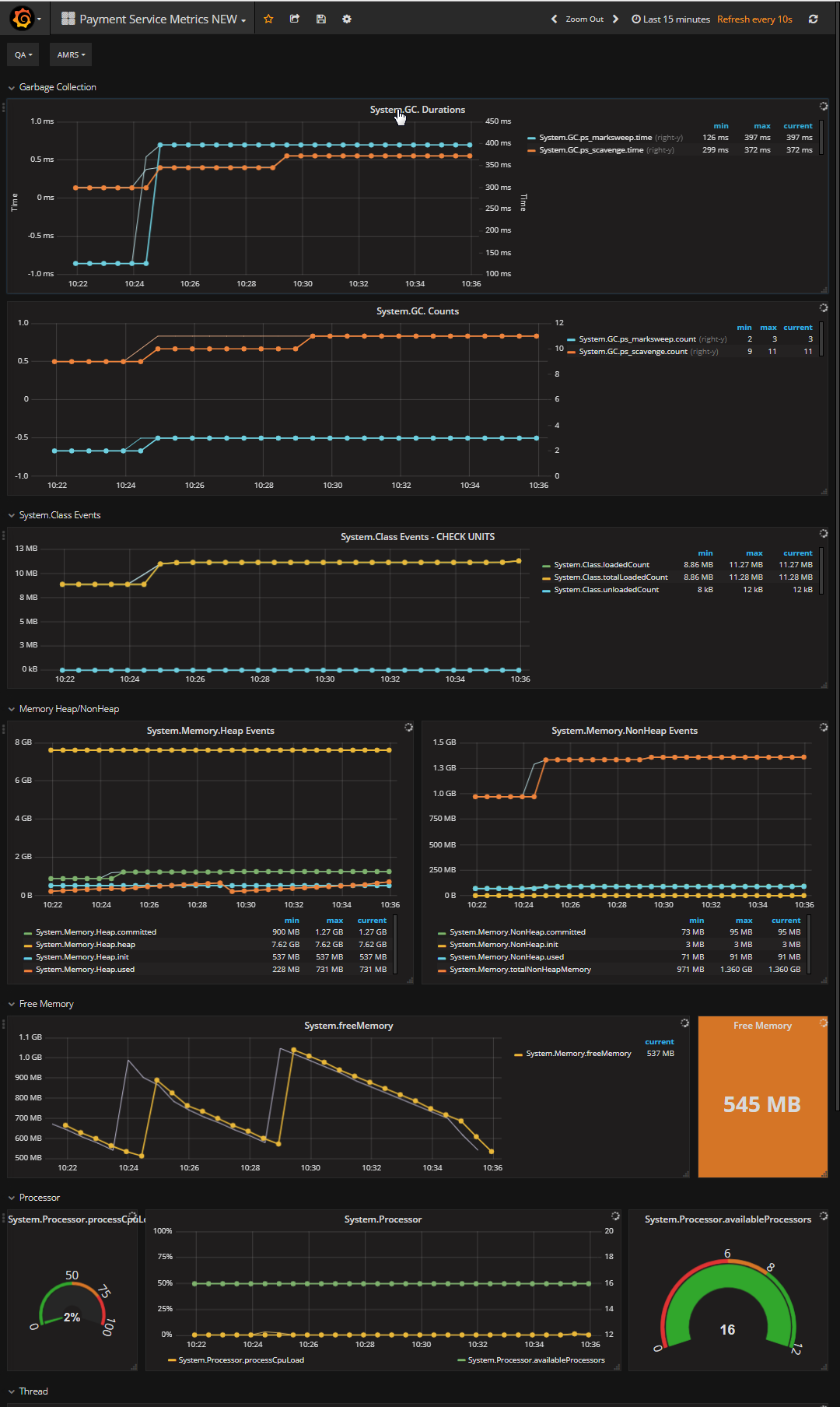
<http://delivery.acm.org.proxy.lib.uwaterloo.ca/10.1145/2900000/2896587/p62-myers.pdf?ip=129.97.58.73&id=2896587&acc=ACTIVE%20SERVICE&key=FD0067F557510FFB%2E9219CF56F73DCF78%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1529996911_202786830f63feb67a6a1f5f05be0530>

## [Beginning JSON](https://proxy.lib.uwaterloo.ca/login?url=http://books.scholarsportal.info.proxy.lib.uwaterloo.ca/viewdoc.html?id=/ebooks/ebooks3/springer/2015-05-26/1/9781484202029)

### Smith, Ben, author.

### Berkeley, CA : Apress 2015

Appendix A – Grafana Dashboard



<https://uwaterloo.ca/math/sites/ca.math/files/uploads/files/tableofcontentstablesfig_2.pdf>

<https://uwaterloo.ca/math/current-undergraduates/co-op-information/work-report-guidelines/appendix-3-work-report-checklist>

<https://owl.english.purdue.edu/owl/resource/560/18/>