

## Conceptual Model of a Dashboard for Monitoring Microservices

Prayudi Utomo<sup>1,\*</sup>, Falahah<sup>2</sup>

<sup>1</sup>Informatics Department, Universitas Widyatama, Bandung, Indonesia

<sup>2</sup>Information System Department, Universitas Telkom, Bandung, Indonesia

### Abstract

The popularity of microservices architecture in building software for the distributed environment leave some problem, such as configuration, orchestration, and monitoring. Some technology and software had been built and implemented, but it has certain constraints for implementation such as its complexity in configuration and needs many components for running. The aim of this research is to propose the conceptual model of a dashboard for monitoring. The model consists of components, the interaction between components, and the requirements for each component. We used Model Driven Architecture (MDA) for describing the model and building the visual prototype based on the model. We used a black-box approach and monitored the microservices through its endpoint. We also propose a simple algorithm for determining endpoint status. We argue that our model will provide flexibility on implementation and can be implemented in a small or medium microservices environment.

**Keywords:** microservices, monitor, model, conceptual, endpoints, MDA.

Received on 07 June 2020, accepted on 27 August 2020, published on 07 September 2020

Copyright © 2020 Prayudi Utomo *et al.*, licensed to EAI. This is an open-access article distributed under the terms of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>), which permits unlimited use, distribution, and reproduction in any medium so long as the original work is properly cited.

doi: 10.4108/eai.7-9-2020.166285

### 1. Introduction

The popularity of microservices technology keeps arising since it appeared on the internet in the middle of 2013 (according to google trends) or as mentioned on James Lewis and Martin Fowler's blog posting [1]. Discussion about microservices is usually often compared with monolithic technology, such as the advantage of microservices in increasing the performance of application dramatically, regardless of the scale of a user on the application. Another benefit of using microservices is flexibility in the development process, both technology and programming language, as long as they can communicate. Many companies are transforming their software architecture from monolithic into microservices. Some popular names such as Netflix, Amazon, eBay, and Soundcloud have promoted the success of microservices implementation [2]

Despite these benefits, there are lots of challenges in implementing microservices that can be categorized into

two groups which are technical and organizational challenges. In technical challenges, we can determine some cases such as testing, infrastructure, integration, logging, and monitoring. [3] The problem of monitoring microservices also becomes a significant finding on the pain of microservices [4]. The problem of monitoring exists on the operation stage from design-development-operation cycle

We also know that there are some tools for monitoring microservices such as Raygun APM, Zipkin, Apache Kafka, Grafana, and Prometheus. The tools mentioned above are usually complex and suitable for monitoring large and complex microservices systems. All the tools need to be configured and usually runs on complex architecture. The dashboard is also too complex so it makes the tools not easy to understand for the novice users for simple monitoring purposes. To resolve this problem, we propose a conceptual model for building the dashboard for monitoring microservices that can run in a simple environment and can be used as a simple tool for monitoring microservices. We used the Model Driven

\*Corresponding author. Email: prayudi.utomo@gmail.com

Architecture (MDA) approach as a modelling tool to help us to define the composition of the model.

The goal of this article is to gather, analyse, and propose the conceptual model of a dashboard for monitoring microservices that were inspired by existing systems and try to address the complexity of system monitoring.

The contribution of this research is explained below:

1. Propose the conceptual and generic model for monitoring microservices.
2. Identify the significant features and process of monitoring.
3. Propose the practical approach of defining the status of microservices through endpoints.

The remainder of the article is structured as follows: Section 2 describes the fundamental concept of microservices architectures and basic concept of monitoring; section 3 presents the research questions and the method for resolving it; section 4 shows the result of the concept; section 5 shows the implementation of the concept; section 6 shows the related works and its difference with the model we proposed and section 7 is the summary and conclusion.

## 2. Microservices Architecture

In this section, we will describe the overview of microservices architecture, its benefits, components, the steps of development and problems that can arise, and how to monitor them.

### 2.1. Overview of Microservices

Microservices is considered a new architecture style on software engineering that consists many small units called microservices which each of them can provide specific services. The microservices solution arose from problems that came from previous application styles, monolithic in nature, which is large, and become complex and not easy to maintain [5]. On a microservices architecture, the modules have to be identified and packaged into a service. The service should be designed in appropriate granularity and implementing microservices needs careful planning and design [6].

The differences between monolithic and microservices architecture can be described based on five aspects [7]: basic architecture principles, scale, database, deployment, and coupled properties. We can see the differences as shown in Table 1.

We can illustrate the architecture of microservices from the perspective of the way of communication and coordination, as explained by Taibe et al. [8]. Generally, the microservices stand behind a “gateway” that can be an API-Gateway, Service Discovery (client-side), or Service Discovery (Server-side). Regardless of the form of the gateway we choose from Taibe et.al, [8] we can conclude that for accessing the microservices, we need to access the gateway first.

Table 1. Differences between Microservices and Monolithic architecture [8].

Aspect	Monolithic architecture	Microservices architecture
Basic principles	Built as a large system and usually one code-base	Built as a small component based on business functionality
Scale	Hard to scale based on demand	Easy to scale based on demand
Database	Uses a shared database	Each module has its database
Deployment	Larger code base using single IDE	Small size code and independent
Coupled	Tightly coupled, so it is difficult to change technology, code language, or framework.	Loosely coupled, easy to change the technology or framework because every module is independent.

The gateway will provide some services such as routing the message, balancing the load, or forwarding the response from microservices. Figure 1 shows the example of accessing microservices using API-Gateway. In the figure, we can see that the API-Gateway becomes a single-entry point for accessing the services. By implementing a single-entry point, it will be easy for us to monitor and control the services.

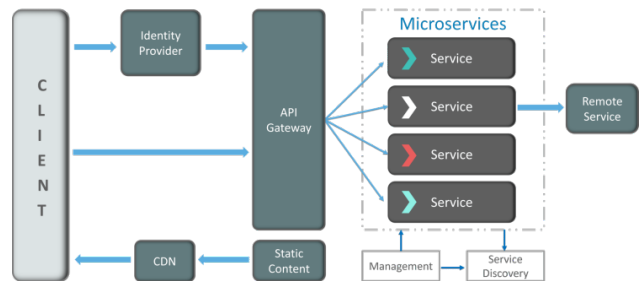


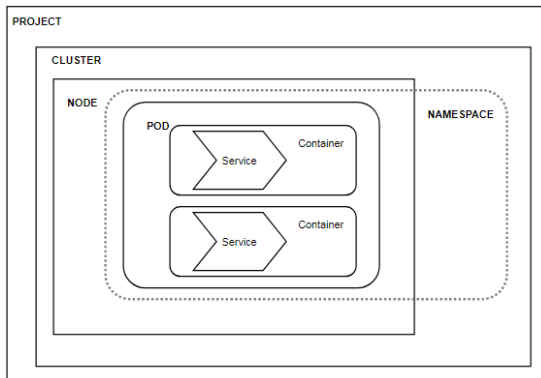
Figure 1. API-Gateway on Microservices [9]

The microservices are built through three phases, which are design, deploy, and observe or operation [10]. The designing phase starts from monolithic, then continues with designing the microservices by defining the scope of services, design the communication between the services, and design for resilience.

In the deploying stage, we can choose some strategies on implementing microservices, which can be divided into 2 areas, with containers or without containers [11]. The issues that needs to be addressed in this stage such as how to standardize artifacts of microservices deployment and implement continuous delivery. In the observing phase, the microservices that are already running need to be

monitored to avoid the failure, and to understand the behaviour that exists across hundreds of services.

The microservices also can be arranged into some containers that run in certain environments. Figure 2 shows the illustration of microservices that run under the container, in which one container will handle one service. The container can be clustered to become a pod, the pod then can be grouped into a node, and the node can also be grouped into clusters. The project is an application-level [12].

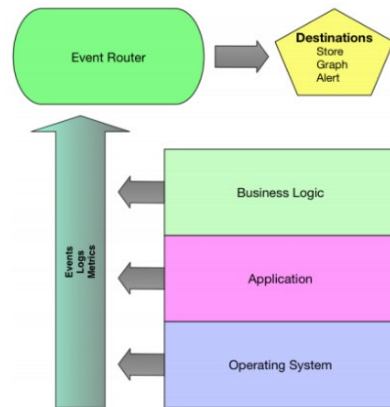


**Figure 2.** Encapsulation of Microservices inside the Container [12]

## 2.2. Monitoring Microservices

Monitoring microservices can be done in several ways. Turnbull [13] defines a general framework for monitoring which consists of two things: events and metrics. Both of them will provide the state of our environment and its performance. Turnbull proposed the framework as shown in figure 3. On Turnbull’s framework, the monitoring process can be done by implementing events, logs, and metrics. The log will record the events based on its metrics and it can be recorded from the targeted system, which also can be divided into 3 layers: business logic, applications, and operating system. Using an event router, this log can be presented on a destination dashboard using a store, graph, or alert.

Based on Turnbull’s framework, we can determine that the component of monitoring consists of the targets to be monitored, the event, logs and metrics, event route, and destination to show the result. Turnbull also mentioned two approaches to monitoring, black-box, and white-box. In the white-box approach, the microservices should be made to be able to monitor, which means, the microservices should have a function for sending the value of the parameter we want to be monitored.



**Figure 3.** Monitoring Framework [13]

Regarding the architecture of microservices, the monitoring process also can be done for each level of microservices layer such as node layer, container layer, application layer, and dependent service layer [14]. Accessing the node layer and container layer needs permission from the system monitored and can be done by inserting the code to explore the value of monitored parameters.

## 3. Study Design

To build the conceptual model of the dashboard for monitoring microservices, we followed the modification of Design Research Methodology [16] for addressing our goal and answering the research questions. The following subsections describe in detail the study design and its execution.

### 3.1. Goal and Research Questions

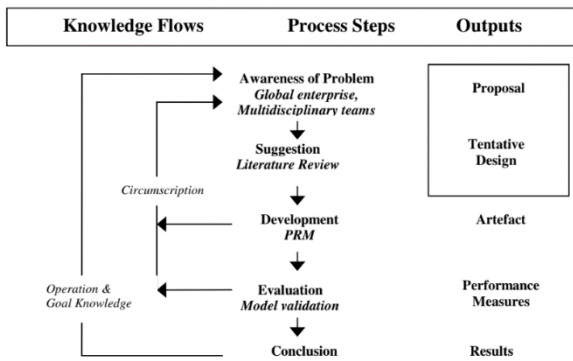
This study’s goal is to propose a simple and easy to implement model of a dashboard for monitoring microservices. We present the model as generic as possible so it can be implemented in any technology and environment. Based on this goal, we defined the following research questions:

1. **RQ1:** What is the generic component for monitoring microservices
2. **RQ2:** What is the simple approach to developing a microservices monitoring system.
3. **RQ3:** How can we present the simple model for monitoring microservices.

### 3.2. Research Methodology

The research was conducted with Design Research Methodology (DRM) [15] that consists of 4 stages: research clarification, a descriptive study I, prescriptive

study, and a descriptive study II. There were some modifications and enhancements of DRM in the field of software engineering, such as making a design cycle [15]. In this study, we adopted the DRM that was modified by Tuffley [16] for building a process reference model (PRM). Tuffley modified the design research methodology making it 5 steps: awareness of the problem, suggestion (literature review), development, evaluation, and conclusion. Figure 4 shows the steps of DRM as proposed by Tuffley.



**Figure 4.** Adaptation of Design Research Methodology in Software Engineering by Tuffley [16]

According to the approach, we implemented the research methodology as follows

1. Awareness of the problem: at this stage, we discussed the importance of monitoring issues in a microservices architecture. We already discussed this topic in the introduction section of this article that mentions the existing monitoring system usually comes with complex interfaces and not suitable for monitoring a simple microservices system.
2. Suggestion or literature review: We discussed basic principles of microservices architecture and general framework for monitoring by Turnbull [13], in section 2 and also discussed some related works on monitoring in section 6. In building the model, we also considered the MDA approach for defining the component in the hierarchical structure.
3. Development: We build the conceptual model of a dashboard for monitoring microservices using the MDA approach and limited the implementation to 2 layers, CIM and PIM. We do not cover the implementation of the concept on PSM because it is out of our scope and it will drive the concept into complexity.
4. Evaluation: We did not use a specific method for evaluation of the model due to some constraints for implementing the model in a real environment. We replaced this topic by showing how to implement the model using simple interfaces. We described the result and future issues in section 5.
5. Conclusion: We summarized the result and its implementation in section 7.

### 3.3. Limitation and constraints

We set some limitations and constraints for our study, based on our goal and research questions, as follow:

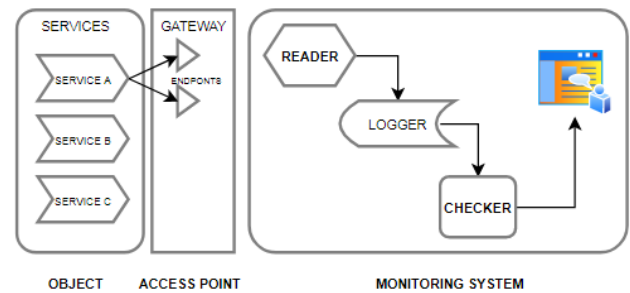
1. The target of monitoring is limited to small to medium scale of microservices, so the model will be suitable for organizations that are in the early stage of microservices implementation.
2. The model focuses on monitoring the performance of microservices and does not include the security aspects.
3. We proposed the model for designing the dashboard, which includes the component that should appear on the dashboard based on minimum requirements of monitoring microservices, and we did not include the detailed mechanism for logging or information retrieval from a log.
4. We used MDA for the modelling approach and limit it to the first two layers: Computational Independent Model (CIM) and Platform Independent Model (PIM).

## 4. Results

In this section, we discuss the result of our study that consists of components we need to identify and the modelling approach.

### 4.1. Components of Monitoring Model

Before we identify the model, we need to identify the component of the monitoring model. Adopting the framework from Turnbull, we proposed a framework to help us identify the component of monitoring, as shown in figure 5.



**Figure 5.** Framework for Monitoring Microservices System

The components of the framework can be defined as follow:

- (i) The object is the thing we want to monitor. In this case, the object is the microservices itself.
- (ii) The access point is the way we access the object to be monitored. In this case, we assume that the microservices are placed beside the gateway and we



could not access them directly, so we chose to access them through endpoints. Let us assume that one microservice can have some endpoints, so we need to address each endpoint to check the availability of the service.

- (iii) The monitoring system is one that can monitor, which consists of several elements such as:
  - a. Reader, the element on the system that can read the parameter value from the object through the access point.
  - b. Logger, the place for storing the parameter value.
  - c. Checker, the element on the system that checks the value from the logger and compares it with a certain standard and sends the result to the dashboard.

Turnbull also defines 3 important components of monitoring, which are event, log, and metric.

Metric is the parameter we want to measure. Metric consists of the parameter and the threshold. The parameter is something we can measure, and the threshold is the minimum quality standard that should be met by the parameters. In our proposed system, we defined the parameters and threshold as below:

- (i) Parameters: We will use four parameters which are: failed requests, server response time, server requests, and availability. The four parameters were chosen based on consideration of easiness on data collecting. The reason why we chose these parameters because we accessed the microservices through endpoints, the we used the four golden signals (Latency, Traffic, Errors, and Saturation) [17] to identify the status of endpoints. Each signal can consist of several parameters.
- (ii) The threshold is the limit value of each parameter we decided to show that the endpoints failed. The threshold depends on the type of parameters, such as seconds for response time, frequency (events per certain period) for failed requests, and server requests.

The log is the place we stored the data. The log can be a text file or database file that arranges the data into a certain format.

The event is the time when we get the value of the parameter. Sample of an event such as a failed event that can be generated from error handling, or alert is generated based on certain criteria (threshold). In this model, we assumed that the reader will collect the data from objects in a certain period, for example, each one second, and store the value in the log.

## 4.2. Modelling Approach

Some aspects need to be considered before defining the model, which are: monitoring mode (black-box vs white-box), level of metric we want to collect (application level),

type of display, and display journey. Table 2 below shows the option for each aspect and the shadowed area is our chosen solution.

Table 2. Options for Design Aspect

Aspect	Options			
Monitoring-mode	Black box		White box	
Level of metric	Node	Container	Application	Dependent services
Type of display	Counter	Gauge	Timer	
Display journey	General		specific	

Monitoring mode has two options, which are black-box and white-box. In this model, we chose black-box since we only accessed the microservices through its endpoint and would not modify the code inside the microservices so it could report itself. On the level of metric, we chose application level since for monitoring the first 2 options (node and container), we needed direct access to the environment where microservices was running. Type of parameter's display: we could implement 2 types which are score metric (number) or gauge. Display journey was chosen because first it would display all selected microservices and then we could choose specific microservices for detailed information. We will use DMM as a short name for the dashboard for monitoring microservices.

We used the MDA approach in defining the conceptual model of DMM. In MDA, the model consists of Computational Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), and Implementation Specific Model (ISM) [18] that can be considered as model layering. The lower the model, the more detailed and specific the result, so it can be said that PSM is more detailed than CIM and PSM is more specific than PIM. According to our research goal, we chose the layer of MDA for describing our conceptual model, which were CIM and PIM. Each layer consists of several processes and the output for each process can be seen in Table 3 below.

Table 3. Process and Output in Modelling Activity

Model	Process	Output
CIM	Define the requirement	List of requirements
	Identify the functions of the system	Use case diagram
	Identify the system architecture	Architectural diagram
	Define activity in a certain function	Activity diagram
PIM	Define system specification	System specification
	Define data/class model	Class diagram

Model	Process	Output
	Define the specific process	pseudocode

### 4.3. Model of Dashboard for Monitoring Microservices

#### A. Computational Independence Model (CIM)

Since the objective of this research was to propose a simple and generic model of DMM, we defined the requirements as follow:

- (i) The system should be able to display the status of microservices based on their end-point condition.
- (ii) The system should be able to display the status of each monitored parameter for each endpoint.
- (iii) The user can customize the period of reporting in the display area.
- (iv) The system can send alerts to a specific address on a specific parameter, as defined by the user.
- (v) The system can produce a report based on user requests.

Based on the list of requirements above, we could draw the use case for DMM as shown in figure 6.

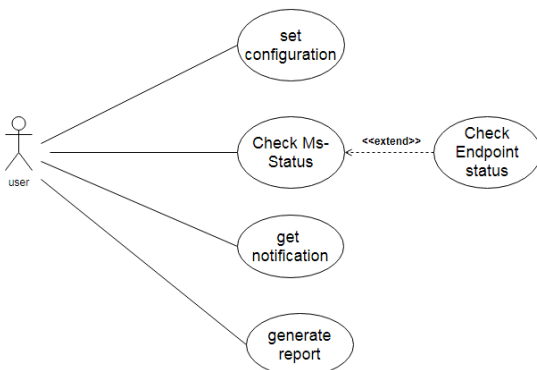


Figure 6. Use Case Diagram of DMM

As we can see from the use case diagram, there are five main features of this system. Set configuration will be used for registering microservices, define for setting the reporting period, and account for sending notification. The feature of check endpoint status will run the checker function because all the output in DMM will depend on this function. We describe the process of check endpoint status using an activity diagram as shown in figure 7.

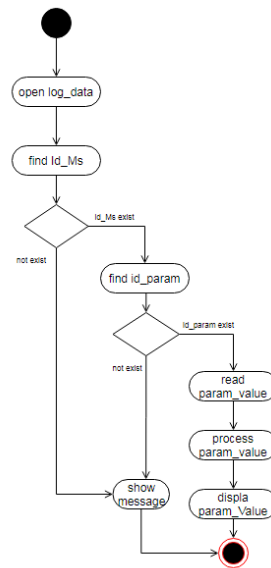


Figure 7. Activity diagram for process check endpoint status

Based on the requirements mentioned above and the activity diagram, we could identify the component of the DMM system using the ArchiMate approach, which are actor, services, application, and its component, data, and technology. Since we were in the CIM stage, we did not include data and technology on the architecture model. The components needed for building the DMM are:

- (i) Actor: User
- (ii) Business services and process:
  - a. Display the status of microservices, get a notification, and produce reports.
  - b. The process for delivering the services above are set configuration, read the log, display the data, send a notification, and create a report.
- (iii) Application component and process:
  - a. The main component of DMM is the dashboard itself. Inside the dashboard, there are some processes such as: setting the configuration, reading the data from log, processing, and displaying the data, setting and sending the alert and generating reports based on user requests.
  - b. The Read process can consist of sub-processes such as: sending requests and recording the response, sending data and recording the result, and so on. The response is recorded on a log file.
- (iv) Data component:
 

The data component for the system is a log file. The log file is generated by the sub-process in reading. The log file consists of a time series of recorded values for each parameter, written by sub-process.
- (v) External component:
 

The external component is the microservices itself which can be accessed using its endpoint. The DMM system will identify the microservices endpoint and register it into the DMM catalogue.

Figure 8 shows the architectural model of DMM.

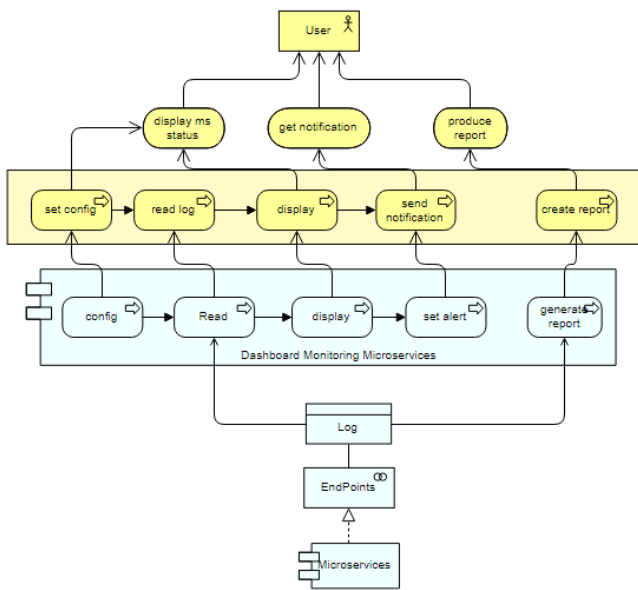


Figure 8. Architectural Model of DMM

**B. Platform Independent Model**

Platform Independent Model describes the specification of the system based on its business requirement. As derived from the previous section, DMM should have some features such as setting the configuration, reading the log, displaying, sending alerts, and generating reports. Based on this requirement, we can define the detail specification of DMM as follows:

DMM should have three main functions which are: create the log, analyse the log, and visualize the log. For each function, we can define the process as shown in Table 3.

Table 3. Process in DMM Function

Function	Process
1. Create the log	Send request directly to an endpoint Accept the response
2. Analyse the log	Write the response to the log file Read the values for each parameter's status on the log Compare the value to threshold values
3. Visualize the status	Set status for each endpoint. Visualize the status for each endpoint by set the colour for each status. Visualize the status for the services by displaying the recapitulation of status for each endpoint that exists on each service

Before the system can perform its function, the user needs to set up some configuration such as:

- (i) Registering the services as an object for a monitor.
- (ii) Setting the period of monitor, or cut-off for reading the log and displaying the result, for example, every 10 minutes.
- (iii) Defining the threshold for each parameter, or using the default value, for example, we can set the threshold of response time to 5 seconds. If the response is more than 5 seconds than we can set the status as fail.
- (iv) Setting the time range for sending the request to the endpoint, for getting the response such as every 5 seconds.
- (v) Setting the account for sending the notification.

As we can derive from the specification above, we can identify the class data for DMM such as service\_catalog, service\_endpoint, parameters, log, notification, and configuration. Figure 9 shows the class diagram for DMM.

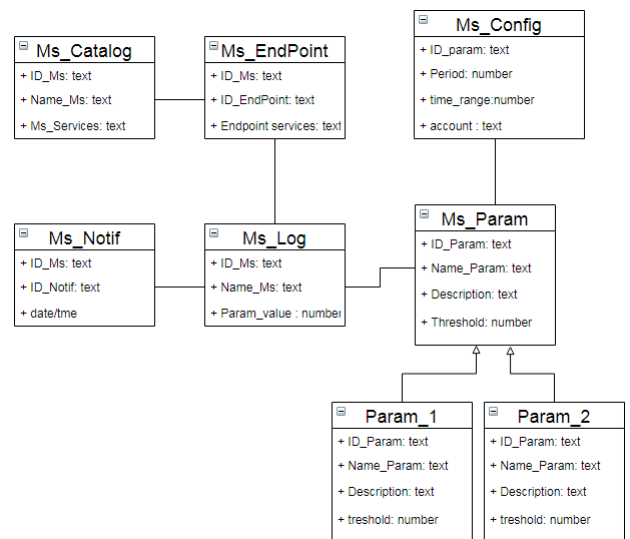


Figure 9. Class Diagram for DMM Model

The main information that will be displayed on the dashboard is the status of the object observed. On our proposed model approach, we determined the status of microservices by exploring the status of endpoints. We chose this approach because the model we built is based on black box monitoring approach, where the monitor system accesses the microservices by accessing the endpoint. One microservice can have lots of endpoints based on their functions, such as to get, post, put, and delete. Failure in one of these endpoints can cause the failure of the microservices. The four parameters on examining the state of the endpoint, are failed response, server response time, server request, and availability. The four parameters we derived from four golden signals for monitoring distributed system which are latency, traffic, error, and saturation [19]. For each parameter as mentioned, we can define the threshold values, which depend on the organization's condition or previous experience of the users. On setting status process, we propose to use the algorithm as shown in pseudocode below. The pseudocode is

written for setting a status of one endpoint for a certain time. So, we defined:

- (i) Four parameters as an array of  $P[n]$ , each for failed response, server response time, server request, and availability,
- (ii) Threshold ( $T$ ) as an array of  $T[1]$ ,  $T[2]$ ,  $T[3]$ , and  $T[4]$ , for each parameter in  $P[n]$ ,
- (iii) The status = Success, Warning, Danger and Off. The criteria for defining status is as follow:
  1. Success: if count of  $(P[n] > T[n]) = 4$
  2. Warning: if count of  $(P[n] < T[n]) = 1$
  3. Danger: if count of  $(P[n] < T[n]) = 2$
  4. Off: if count of  $(P[n] < T[n]) = 4$

Based on assumption above, we can define the pseudocode for defining status as below:

```
// Value of Parameter P[n] on certain time t
var P = [];

// Threshold
var T = [];

// Counter
var count = 0;

checkCount() {
    P = readLog(); //reading the log file
    for (i = 1; i <= 4; i++) {
        if (P[i] > T[i]) {
            count = count + 1;
        }
    }
    return count;
}

setStatus(count) {
    switch (count) {
        case (count == 4):
            return "Success";
        case (count == 3):
            return "Warning";
        case (count == 2):
            return "Danger";
        default:
            return "Off";
    }
}
```

## 5. Implementation and Discussion

### 5.1. Interface Design

To show that the model can work, we implement it in a prototype for small e-commerce site that consist 4 services: user, post, product and analytic. We pick the name of our DMM model is Monitorifix. Figure 10 shows the main menu of the Monitorifix DMM model.

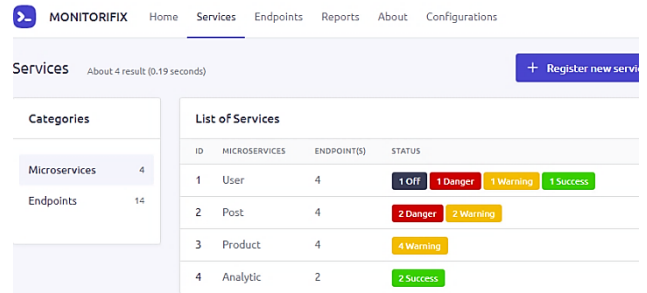


Figure 10. Main Display of Dashboard for Monitoring Microservices

The dashboard will display the categories of services on the left bar and a list of services including the recap of endpoint status in the right bar. The system will have 4 main menus which are services, endpoint, reports, and configuration. Figure 11 displays the list of endpoints that are available in services we monitored. It gives us an overview of all endpoints and the user can choose to display a certain status or all statuses.

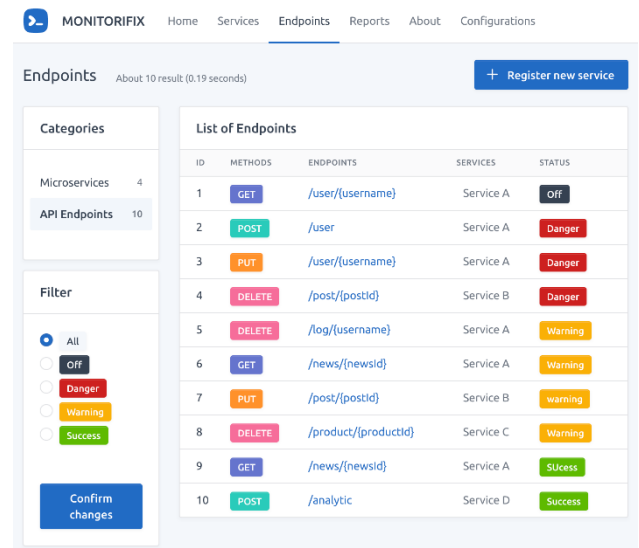
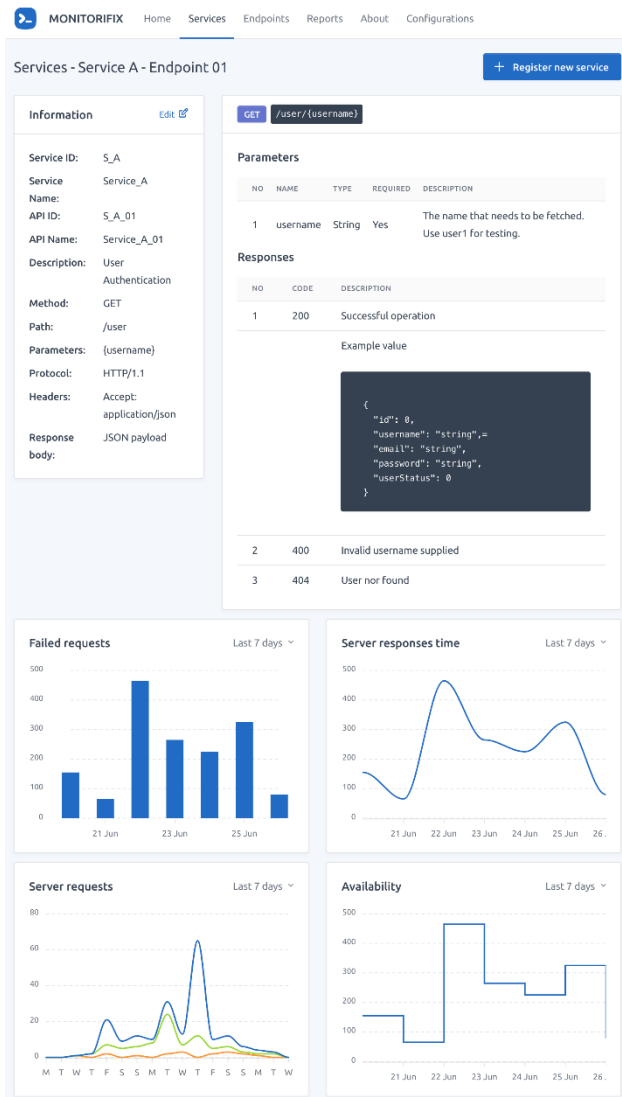


Figure 11. The status of Endpoints

Other features that are available on Monitorifix are report and configuration. The report can be used for generating custom reports such as the status of certain endpoints in a certain period, or comparing some services in one sheet, and much more. Monitorifix also provides a detailed status for each parameter for one endpoint as shown in figure 12. The status will be displayed based on the time period chosen by the user in the configuration menu.





**Figure 12.** The four parameters status of one endpoint

The model is designed as simple as possible and can be implemented in any environment. The interface can be integrated with any tools for generating data or reading the parameter values.

## 5.2. Specific Platform for implementation

There are many possible combinations of technologies for implementing PIM into PSM. We can convert the PIM into certain technology stacks, for example, front-end: HTML/JavaScript, back-end: PHP/Python and Apache web server, data storage: file-based, message broker, database file. Another example of technology stack: front-end: Java, back-end: Java/PHP, data-storage: file-based.

The options of technology stack can be customized depends on the environment capability of supporting the monitoring systems.

## 5.3. Discussion and future opportunities

The DMM model is suitable to be applied in a small or medium microservices environment and focuses on profiling services as a single unit. Although it seems so simple, the dashboard can help the user and developer in monitoring the services without setting lots of configuration that might not be necessary to set for the early stage of microservices implementation. Users can add and remove certain microservices and it can give the user the scalability of monitoring systems.

As mentioned in the previous section, the ability for monitoring systems supported by microservices architecture is a mandatory requirement. The Monitoring system can give us a warning and take important action to keep business services running. We can monitor the system easily if the system has the ability for reporting itself periodically and automatically, by creating a log of its activities. We can implement a good logging system if we can define the most critical parameters to be recorded. It indicates that we need to consider the monitoring process since the beginning process of the analysis and design of a system that runs is based on a microservices architecture. It means that we have to define whether the microservices can report themselves or we need to install a specific component for monitoring and logging.

Along with the growth of microservices in the organization, it will face some critical issues according to monitoring such as:

- (i) **Portability:** the monitor should be independent of the microservice, so it will lead to an idea of a detachable component for monitoring.
- (ii) **Interoperability:** the monitor can run in any certain technology stack, and can support a variety of components and types of microservices.
- (iii) **The complexity of the system:** It would be better for organizations to prepare their strategy on adopting microservices, together with monitoring strategy so they can provide the services and at the same time keep the performance of the services.
- (iv) **The needs of closer monitoring:** a sensor for monitoring can be attached to the microservices component for closer monitoring, and it can give the microservices additional load. There is a consideration about the trade-off on each approach, such as setting the priority of microservices, and it can lead us to choose what strategy we need to implement for logging and monitoring.
- (v) **System security:** The more complex the system, the more issues of security need to be addressed. The microservices that can be built by the different groups need standards on security requirements and protocols, and also need a way to monitor security.

- The next step after we can monitor the performance, is to monitor the security of the systems.
- (vi) Failure preparation: We can set some warnings for parameters to define the failed status of the system and it can be part of an early warning system. The information on the dashboard can help users in anticipating the failure and implement a certain strategy for recovery, such as failover or graceful degradation.

Future opportunities in this research are to identify other parameters for monitoring endpoints and how to set the status of microservices based on endpoints or any parameter.

### 6. Related Works on Monitoring

The issue of monitoring microservices is covered in some related works such as the needs of control loops for a runtime that includes monitoring, analysis, planning, and execution [20]. The previous works on monitoring microservices are varied from using a specific software, proposed dashboard model or proposed model for decision guide in microservices system. We can summarize the comparison of related works in monitoring with our work in table 4 below.

Table 4. Related Works on Monitoring Microservices

Articles	Key point	Comparing with our works
Monitoring and Analysis of Microservices Performance [21]	Uses Keiker framework, collect data and store it locally	Utilizes the log data that can be generated and stored anywhere, no need additional software.
A Dashboard for Microservice Monitoring and Management [22]	Emphasizes the discussion on component information that should exist on the dashboard, not including the approach for gathering the information	Proposed black-box approach on monitoring and the mechanism how to gather information from existing microservices
Decision Guidance Models for Microservices – Service Discovery and Fault Tolerance	Proposed a decision model for services discovery, service registration, versioning, caching, load balancing, and fault tolerance. No decision model for assigning the	Focusing on runtime monitoring the microservices, and emphasizing on service availability by checking the end-point periodically, and provide the preview of status microservices

Articles	Key point	Comparing with our works
Performance Engineering for Microservices: Research Challenges and Directions [24]	status of the services identified three key areas for monitoring issue in microservices, which are: instrumentation for distribution monitoring, measurement, and anomaly detection techniques.	We adopt some concept of architecture for monitoring microservices which are distributed monitoring (we recommend to read the signal from end-point), and additional measure (we recommend the mechanism to set the status of microservices).

### 7. Conclusion

In this research, we proposed a simple model for building a dashboard for monitoring microservices. We proposed the components of a monitoring framework that consists of an object to be monitored, access point, and the system monitoring itself that contains the reader, the logger, and the checker. The system can access the microservices that are usually placed behind the gateway, through its endpoints. The status of each endpoint can be used for showing the status of services. We use four parameters for detecting the endpoint condition and generated the status of endpoint based on the combination status of each parameter compared with its threshold. The system was modelled using the MDA approach and implemented as an interface prototyping using static webpages. The monitoring model we proposed focuses on profiling the status and condition of each service and does not provide information about dependent services. It uses a black-box approach so the system can access all the microservices as is, regardless of the inner structure of microservices, as long as the microservices have the endpoint to access.

### References

- [1] Lewis, J., & Fowler. M., “Microservices” Available: <https://martinfowler.com/articles/microservices.html> (last accessed: 05/05/20)
- [2] Richardson, C., “Microservices Architecture” Available: <https://microservices.io/articles/whoisusingmicroservices.html> (last accessed: 05/05/20)
- [3] Makilato, N, & Mikkonen, T. Challenges when moving from monolithic to Microservices Architecture, in book *Current trends in Web engineering*, pp 32-47.
- [4] Soldani, J, & Tamburri, D.A. The Pains and Gains of Microservices: A Systematic Grey Literature Review, *Journal of System and Software*, September 2018
- [5] Fritzs, J, Bogner, J., Zimmermann, A., & Wagner, S. From monolith to microservices: A classification of

- refactoring approaches, in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (J.-M. Bruel, M. Mazzara, and B. Meyer, eds.), (Cham), pp. 128–141, Springer International Publishing, 2019
- [6] Zhelev, S., & Rozeva, A. Using Microservices and Event-Driven Architecture for Big Data Stream Processing, in *Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics* (AMEE'19), 2019
- [7] Parahar, M. Difference between monolithic and microservices architecture. Available: <https://www.tutorialspoint.com/difference-between-monolithic-and-microservices-architecture> (last accessed: 05/05/20)
- [8] Taibi, D., Lenarduzzi, V., & Pahl, C. Architectural Patterns for Microservices: A Systematic Mapping Study. In *Proceeding of 8th International Conference on Cloud Computing and Services Science, CLOSER (2018)*
- [9] Image is taken from <https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a> (last accessed: 05/05/20)
- [10] Bruce, M., & Pereira, P.A. *Microservices in Action*, Manning Publication, 1<sup>st</sup> ed, 2018.
- [11] Mahajan, A. *Microservices Without Containers*, Available online on <https://dzone.com/articles/microservices-without-containers>, (last accessed: 10/05/20)
- [12] Richardson, C. *Choosing a Microservices Deployment Strategy*, 2018, Available online on <https://www.nginx.com/blog/deploying-microservices/>, (last accessed: 15/05/20)
- [13] Turnbull, J. (2016). *The Art of Monitoring*
- [14] Saito, H, Lee, H.C., & Wu, C-Y. *DevOps with Kubernetes: Accelerating software delivery with container orchestrators*, Packt Publishing (October 16, 2017)
- [15] Glass, R.L, Vessey, I., and Ramesh, V.(2002). Research in Software Engineering: an analysis of the literature, *Information and Software Technology*, 44. Pp.491-506.
- [16] Tuffley, D.J. A Design Research approach to developing a Process Reference Model for leadership of integrated teams in virtual environments, *Ph.D. Thesis, Griffith University, Queensland, Australia*, 2010. Available online on: [https://research-repository.griffith.edu.au/bitstream/handle/10072/365310/Tuffley\\_2010\\_02Thesis.pdf?sequence=1&isAllowed=y](https://research-repository.griffith.edu.au/bitstream/handle/10072/365310/Tuffley_2010_02Thesis.pdf?sequence=1&isAllowed=y), (last access: May 20, 2020)
- [17] Beyer, B., Jones, C., Petoff, J., & Murphy, N.R. *Site Reliability Engineering: How Google Runs Production Systems*, O'Reilly Media, 2016
- [18] Bajovs, A., Nikiforova, O., & Sejans, J. Code Generation from UML Model: State of the Art and Practical Implications, *Applied Computer System*, vol. 14, 2013
- [19] Allclair, T., & Kaczorowski, M., “Exploring container security: Isolation at different layers of the Kubernetes stack” Available: <https://cloud.google.com/blog/products/gcp/exploring-container-security-isolation-at-different-layers-of-the-kubernetes-stack> (last accessed: 05/05/20)
- [20] Garriga M. Towards a Taxonomy of Microservices Architectures. In: Cerone A., Roveri M. (eds) *Software Engineering and Formal Methods. SEFM 2017*. Lecture Notes in Computer Science, vol 10729. Springer, Cham, 2018
- [21] Barakat, S. Monitoring and Analysis of Microservices Performance, *Journal of Computer Science and Control System*, Vol.10 No. 1, May 2017 pp 19-22
- [22] Mayer, B., & Weinreich, R. A Dashboard for Microservice Monitoring and Management, In *Proceeding of 2017 IEEE International Conference on Software Architecture Workshops*,
- [23] Haselbock, S., Weinreich, R., & Buchgeher, G. Decision Guidance Models for Microservices – Service Discovery and Fault Tolerance, in *Proceeding of ECBS '17, August 31-September 1, 2017*, Larnaca, Cyprus
- [24] Heinrich, R., Hoorn, A., Knoche, H., Li, F., Lwakatara, L.E., Pahl, C., Schulte, S., & Wettinger, J. (2017). Performance Engineering for Microservices: Research Challenges and Directions, in *Proceeding of ICPE '17 Companion*, April 22-26, 2017, L'Aquila, Italy