Intelligent Auto – Scalar for Micro services

Design Document

Project ID: 17-021

Author's Name: G.Sinthujan
Author's ID: IT14056826

…….
Dr. Dharshana Kasthurirathne (Supervisor)

Bachelor of Science (Special Honors) in Information Technology
Sri Lanka Institute of Information Technology

May, 2017

Intelligent auto scaler for micro services.

DECLARATION

I hereby declare that the system requirement specification document entitled "Intelligent Auto – Scaler for micro services" submitted to the Sri Lanka Institute of Information Technology is a record of the original work done by myself under the guidance of Dr. Dharshana Kasthurirathne, Project Supervisor, and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

G Sinthujan : …….

Date : …….

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to give a detailed description of the proposed project "Automatic Scaler for micro services" system. It will also define the purpose and declaration on how the system will be developed for optimizing the usage of micro services being deployed on to the cloud platform. This document is primarily intended for the approval of the supervisor and as a reference for developing the system.

1.2 Scope

The proposed system will be used for getting the full benefits of an architecture which was derived from the SOA which is a messaging system. The architecture that will be the base is known as the micro service architecture. The main purpose behind this project is to implement a way in which it would be easier to deploy and manage systems developed using the micro service architecture. This application will be used as an extension to the existing applications which are used for the deployment and management of the micro services.

The components that will be developed for the system are as follows: -

- Usage pattern analysis.
- Resource Monitoring and execution plan generation.
- Decision making and instance scaler.
- Business scenario predictions.

The component that will be covered in detail out of the overall system is the Resource Monitoring and execution plan generation component.

1.3 Definitions, Acronyms, and Abbreviations

Abbreviation	Meaning
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SOA	Service Oriented Architecture
HR	Human Resources
CLI	Command Line Interface
TCP	Transmission Control Protocol
IP	Internet Protocol

Table 1: Definitions and Acronyms

1.4 Overview

Over the past two decades, the improvements to technology that help an individual's day to day life has been vast. This has brought forth a dependency on technology that was not seen in the history of mankind since its birth. The improvements that are made can be said to be miniscule, but when the whole system is considered it shows a major difference.

When considering this, the invention of the computer during the early 1950's can be said to be the biggest break through which has led to the advancement of many other fields. The development of the computer system led to research being opened on a completely new field which is now known as computer science. This is a field that researches on topics which would increase the performance of existing systems while not reducing the efficiencies of existing systems. One such branch that is researched and continuously developed on are architectures. Architectures are basically designs which are used to implement systems which are highly maintainable, efficient and will try to give the maximum performance possible. There exist sub branches for said architectures for example, design patterns etc. In the many available deployments for systems, based on the environment it is being deployed on there exist multiple architectures. We will be looking into an architecture which has made it easy for systems to be interoperable, and which is the oldest to date for introducing message passing mechanisms between different system.

The architecture is known as the *Service Oriented Architecture[SOA]*. Over the years many architectures have emerged that have boasted advantages over SOA, but they still had other drawbacks. One such architecture is known as the ***Micro Service Architecture*** that was introduced in the late 2014's**.** The micro service architecture suggests that rather than having all the required services bundled into a single component, to break down the component into multiple smaller components which are highly cohesive, that is they are particularly focused on one main task. By doing so it makes the system more maintainable and manageable rather than having to deal with the bulky system that will need to be deployed if it were built using the SOA. By the Canonical definition given by Martin Fowler, the initiator of the architecture,

"In short, the micro service architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies."

Even with the many advantages that this architecture provides, many companies are reluctant to use it because as the system becomes more complex, the management of the system becomes a very difficult task.

The main focus of this project is to develop a system that will act as an extension to the Docker [Further explanation will be given below in section 2] which will be able to automate all the mandatory processes, that are required for a system developed using the micro service architecture to function with maximum efficiency while reducing the amount of resources that needs to be allocated to the system. By doing so the orchestration of the system will be partially automated, thus the requirement for having HR manage the services after they are deployed will be removed, while at the same time the management of the resources for the system will also be automated thus making it more efficient and less costly. The users of the system will be companies who are using the micro services architecture while facing the difficulty of managing the continuously growing complexity of the developed application.

The content of this document includes three chapters and appendixes. The second chapter provides an overview of the system functionality and system interaction with other systems. This chapter also introduces different types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.

The third chapter provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are used in order to specify the requirements more precisely for different audiences.

The fourth chapter deals with the prioritization of the requirements. It includes a motivation for the chosen prioritization methods and discusses why other alternatives were not chosen.

The Appendixes in the end of the document include the all results of the requirement prioritization and a release plan based on them.

2 OVERALL DESCRIPTIONS

This section of the document will be used to expand on the overview of the functionality of the system. The main component that will be explained will be the component to be developed by the author of this document. The other components that are necessary for the system to function will be touched on briefly. It will also describe how the said components will interact with one another and explain on the functionality of the component. The other section of this chapter will be to define the constraints and the assumptions that are taken for this system to be built.

2.1 Product Perspective

This system will mainly consist of the implementation of an extension of an existing system known as the Docker. The Docker is basically a deployment platform which is used to deploy containerized systems. The system which we will be considering for our project are systems that are developed using micro services. The extension will consist of a component that will be deployed as a micro service to the system and another component which will consist of all the components from the other members of the group that will act as the main interface for the proposed system. This component will be the main part of the proposed system, as it will be the one that will directly interact with the Docker for making the necessary decision for the management of the deployed applications.

The main interface which will called as the auto-scaler from this point forth, will be able to communicate with the Docker using the CLI. The micro service deployed as part of the system which will be known as the Resource Monitoring Application from this point forth will be deployed on all Dockers which will be using our proposed system. Further explanations on these components will be given below.

As this project is also focused on the analysis of data that is collected, there is a Big-Data analysis component present. This data analysis will take place within the component proposed and the data collected will be stored on to a dedicated storage available within the server, where the system is deployed. The setup of the system will be explained further later. The proposed database for the collection of data, will be a No-SQL Database. The proposed data will be used for storing data from: -

- Usage analysis data.
- Resource monitoring data
- Executed Business scenarios.

The structure below gives the general layout of how the Docker manages its containers.

Container 1	***Container 2***	***Container 3***
Tomcat	*SQL Server*	
Java	*.Net*	*Static Binary*
Debian	*Ubuntu*	*Alpine*
Kernel		
Server Hardware		

Table 2: Docker Component Structure

The problem with the existing tools/plugins available for the Docker such as 'Kubernetes' which acts as a deployment manager and orchestration tool for the Docker is that it does not consider the long term usage of resources that are allocated to the system and rather manages it based on a rule based system. Thus they are not dynamic. The proposed extension will be able to overcome this disadvantage and also introduce the possibility for the system to be scaled vertically.

The main software interfaces for the system include the following

2.2.2.1 Micro Service Resource Monitoring

This is an interface that will be provided with the main purpose of collecting data from each individual micro service instances. After the data has been collected, the data will be transformed in the model builder component and this will be used for retrieving result sets that are necessary for the system to function.

2.2.2.2 Model Builder Component

The model builder component is mainly responsible for the management of unstructured data and the creation of result set that is the building of the derived data set for prediction purposes.

2.2.2.3 Docker CLI Component

This component will be used for the retrieval of the resource allocations for each identified micro service when a particular business plan is being executed.

2.1.2 Hardware Interfaces

Since this application will be designed on a software level, the only interaction with the hardware will be through the kernel of the deployed system. In this context the deployed system would either be the Docker that is deployed as an OS for the hardware or the OS on top of which the Docker is currently operating. It is necessary to have interaction with the hardware to get an accurate readout for the resources that are currently being used by the deployed system which is the containerized application that is currently running in the Docker environment. The minimum required hardware configuration for the system to run on are: -

1. A processor with minimum 4 cores.
2. 4.00 GB of RAM
3. 3.00 GB of available disk space
4. A static IP address

2.1.3 Software Interfaces

As the proposed system is an extension for the Docker, the software interface will be the full installation of the Docker system on to minimal required system requirements. While this is also part the following ports will be needed to be kept open or the configuration should be changed for the Docker to access other ports wherever

necessary. The following table provides the necessary information for the ports that need to be open for the Docker to function.

Hosts	Direction	Port	Purpose
managers, workers	in	TCP 443 (configurable)	Port for the UCP web UI and API
managers	in	TCP 2376 (configurable)	Port for the Docker Swarm manager. Used for backwards compatibility
managers, workers	in	TCP 2377 (configurable)	Port for communication between swarm nodes
managers, workers	in, out	UDP 4789	Port for overlay networking
managers, workers	in, out	TCP, UDP 7946	Port for gossip-based clustering
managers, workers	in	TCP 12376	Port for a TLS proxy that provides access to UCP, Docker Engine, and Docker Swarm
managers	in	TCP 12379	Port for internal node configuration, cluster configuration, and HA
managers	in	TCP 12380	Port for internal node configuration, cluster configuration, and HA
managers	in	TCP 12381	Port for the certificate authority
managers	in	TCP 12382	Port for the UCP certificate authority
managers	in	TCP 12383	Port for the authentication storage backend

managers	in	TCP 12384	Port for the authentication storage backend for replication across managers
managers	in	TCP 12385	Port for the authentication service API
managers	in	TCP 12386	Port for the authentication worker
managers	in	TCP 12387	Port for the metrics service

Table 3: List of Open Ports for Docker

2.1.4 Communication Interfaces

The communication between the different parts of the system is important since they depend on each other. Where the main components that will be doing the communication will be entirely isolated from the other components of the Docker thus it will only be dependent on the Docker for certain operation. All other communication will be between the sub components that will be present in the proposed system. The following will be the ways of communication between the Docker and the proposed system.

- Service for accessing CLI of Docker
- Service from the deployed micro service for collecting data from the resource monitoring module.
- Service for providing required reports.

2.1.5 Memory Constraints

The only memory constraints that will be present for the system will be the server's configured hardware.

2.1.6 Operations

This section will be used to explain the process of installing the proposed system [Resource monitoring component].

- Install the Docker.
- Configure the Docker system.
- Deploy the resource monitoring micro service module.
- Include the middleware necessary for monitoring the micro service.
- Ensure that the port for communication with service is setup.

2.2 Product Functions

The product will consist of 4 main components which are: -

1. Web API usage pattern analyzing module
2. Absolute resource utilization analyzing and micro service tagging module
3. Business scenario prediction and analytics visualization module
4. Micro service instance managing and decision making module

The main component architecture diagram is given below

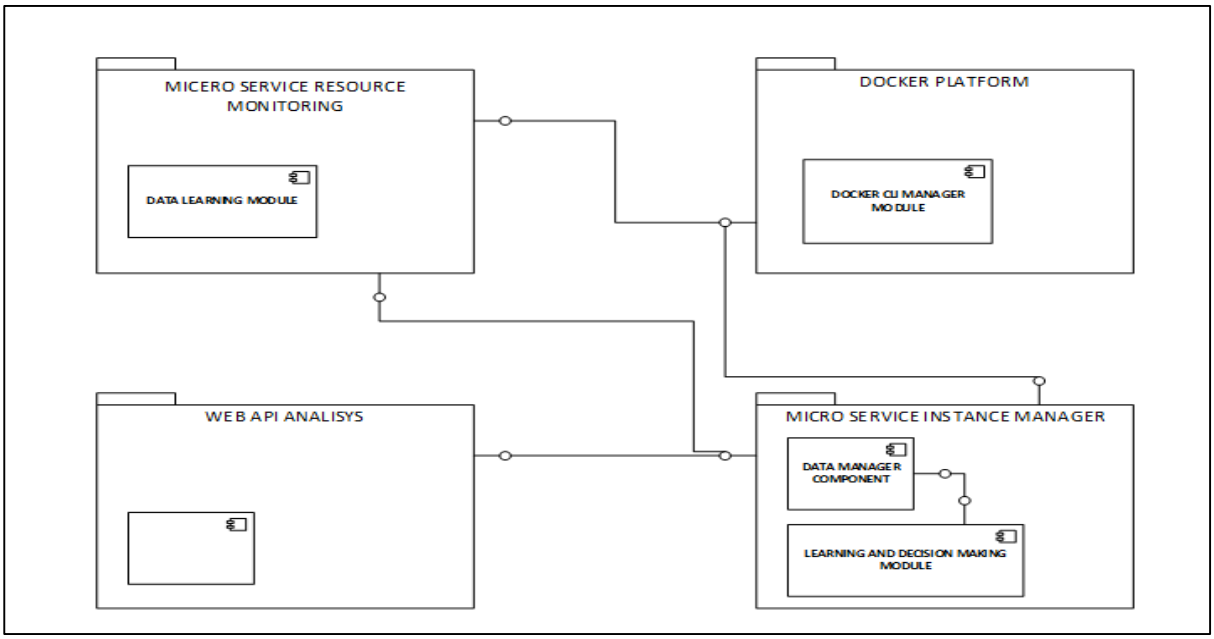

Figure 1: Component Architecture Diagram

The Web API usage will be the entry point for the system. This module will be monitoring the requests that are made to the system and make a record. The data collected will be used for predictions.

The next component would be the resource utilization analyzing module and the micro service tagging module. The resource utilization module will be placed as two components one which will be outside the influence of the Docker another a micro service which will be deployed on the system to get the data from the individual micro service. For this purpose, a middleware will be developed which will be included as a library for each individual micro service deployed onto the system. The main functionality that will be carried out by this middleware is to uniquely identify each micro service and to uniquely identify the task that the micro service is currently executing.

The main purpose for doing so is to uniquely get the resources which are being used up when a service call is made and the micro service is executed. The other reason for

this is identify the currently executing micro service sequence, that is the orchestrated scenario of execution when a particular service is called. The middleware will redirect the execution time along with the resources being used to a service that will be custom deployed and will only have the task of streaming the data being fed from the other services to an interface which will connect to the component which will act as the data analyzer and decision maker.

The resource monitoring micro service will be deployed independently of the other services. This module not require a service discovery mechanism as the registration of services will be dynamic. After the service has been discovered, the service will be registered and the streaming of data will commence. The basic architecture of the micro service is given below.

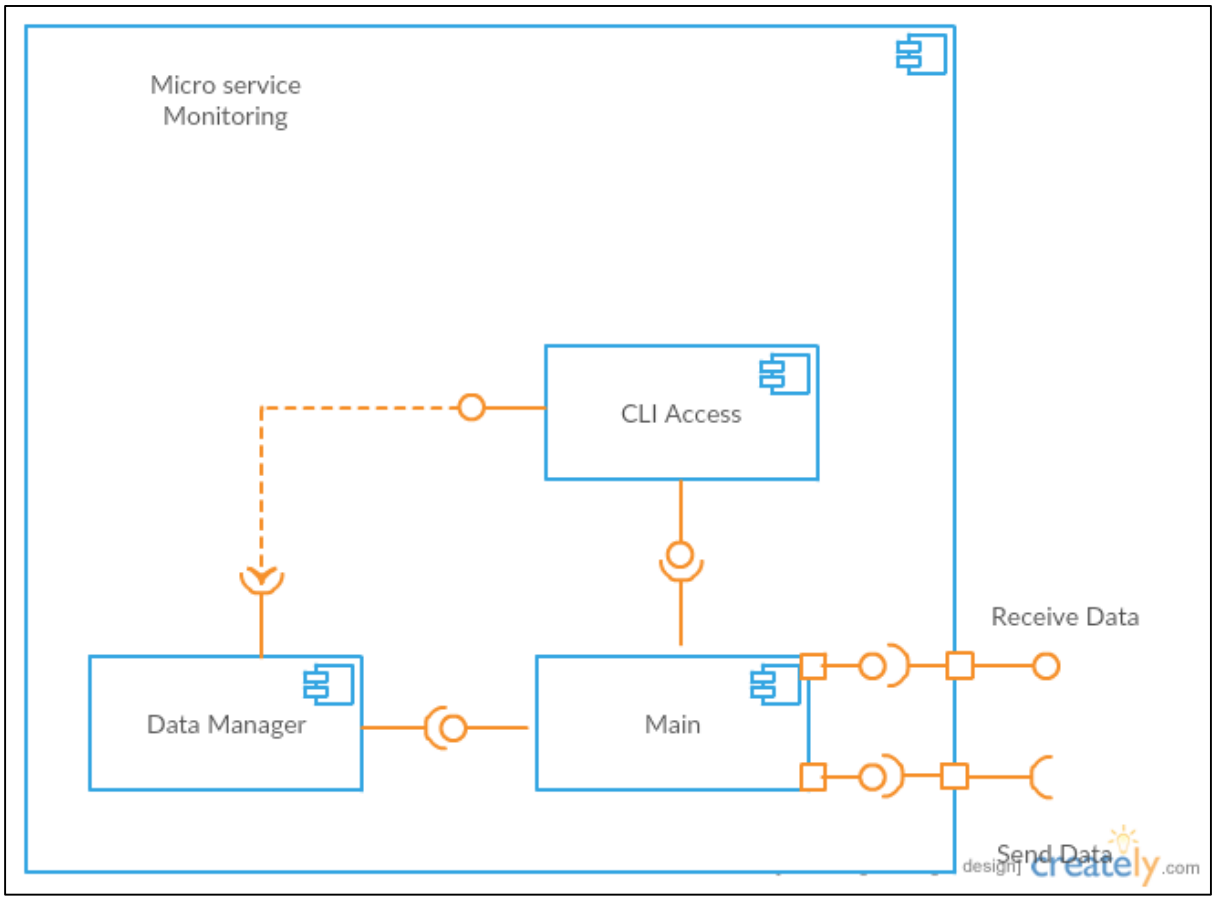

Figure 2: Sub Component Architecture for Resource Monitoring Service

The Resource monitoring micro service will comprise of the Data Manager which will be responsible for collecting the required resources data from the CLI of the Docker and formatting it to the necessary format before sending it to the learning module component, which will be placed outside the Docker system. The CLI Access component will be responsible for accessing the interface of the Docker and executing the necessary commands. This service will also make use by the instance manager for creating and destroying the containers available in the Docker based on its decisions. It will also be responsible for the generation of the business execution plan when a particular service is called. The sequence will be initialized when a service is called

and will be built on the service that are called by the orchestration. An example of how the services will communicate with the Resource monitoring module is given below

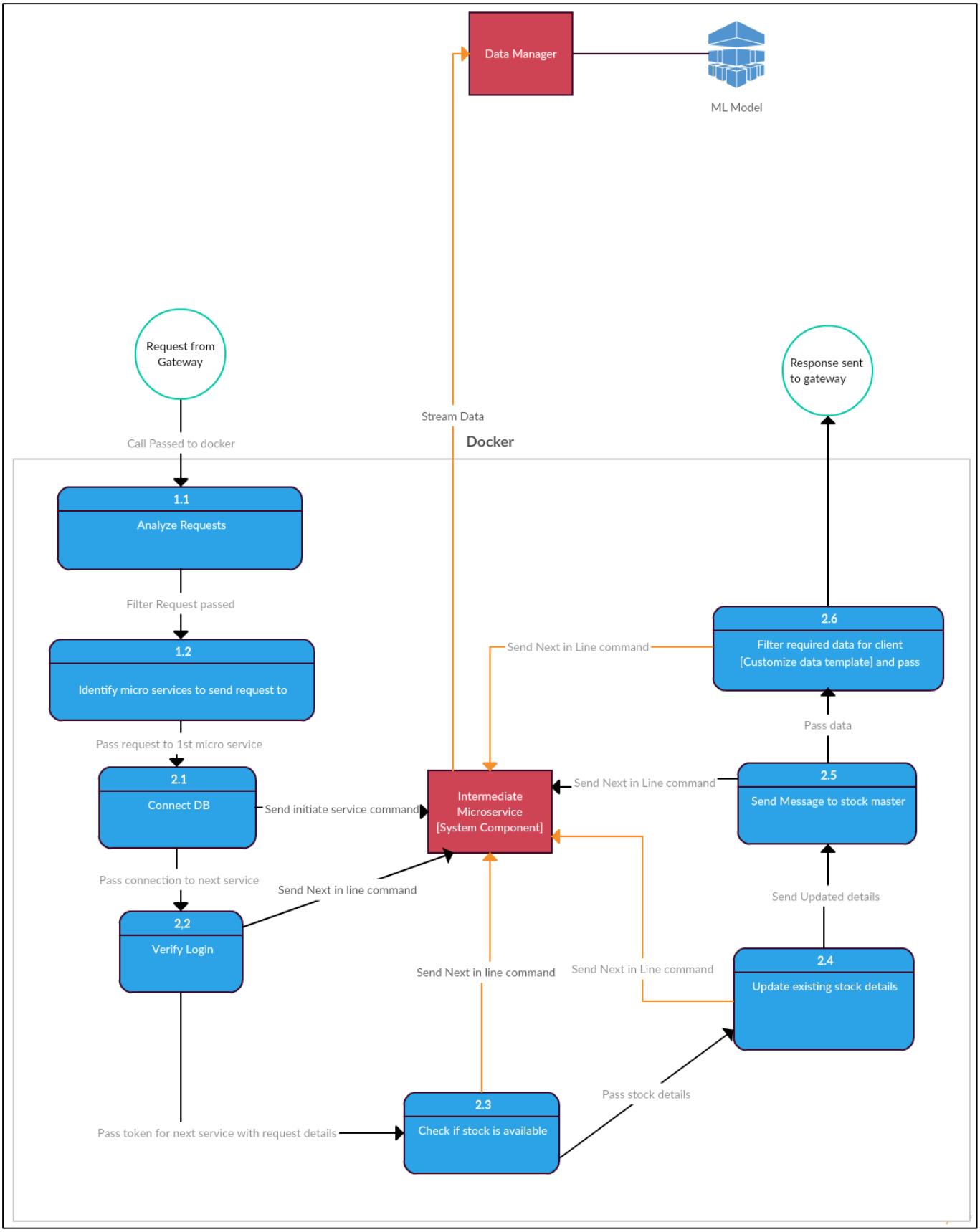

Figure 3: Example of sequence identification for resource monitoring

The above given scenario is for a stock managing application. The business scenario that is executed in the above call is for a stock update. As the diagram denotes, for every call initialization the payload is sent to the micro service for preparation for collecting data. At the end of this it is terminated and the next services details are called. As there is not much processing overhead for the micro service as a whole, the need for managing it will not be necessary. But as a precaution the instance manager will be monitoring this service as well.

Once the necessary data has been collected, the learning model for the resource allocation will need to be created, this will be used by the instance manager for making decisions. At the initial stage when the system has been integrated to the Docker, the learning process will be supervised for a pre-defined period of time to ensure the reduction in errors. But after the training period is over the system will be trained by making use of the unsupervised learning model. The data that will be given off this model include

1. Predictions of the individual micro services resources.
2. Predictions of the application resource usage.

Web services will be developed for accessing the result set from the model. The Business execution plan will be collected and stored as an un-structured dataset and this will be given to the 3rd component for analysis.

The 3rd component's main purpose is to analyze the data provided by the 2nd component and to predict the services that are highly required, predict the next executed business scenario against time, and to finally provide the data for visual representation by the administrator of the Docker system. The administrator will be able to view the data feed from each individual micro service and the application as a whole, along with the request patterns variations in a report manner.

The 4th and final component is the core of the system. This component will take into account all the data being fed from the other components and decide on the management of the containers of the Docker, along with which it performs other tasks such as ensuring the availability of the most required micro services.

2.3 User Characteristics

The auto scaler is an automated system, with the initial setup and configuration which will be done by the administrator of the Docker system or the network administrator who manages the server.

2.4 Constraints

Research constraints limit the functionality of the system. This would include the system policies defined for the server, memory constraints, dependent

application’s compatibility, data communications constraint [Firewall constraints etc.].

2.5 Sequence Diagram

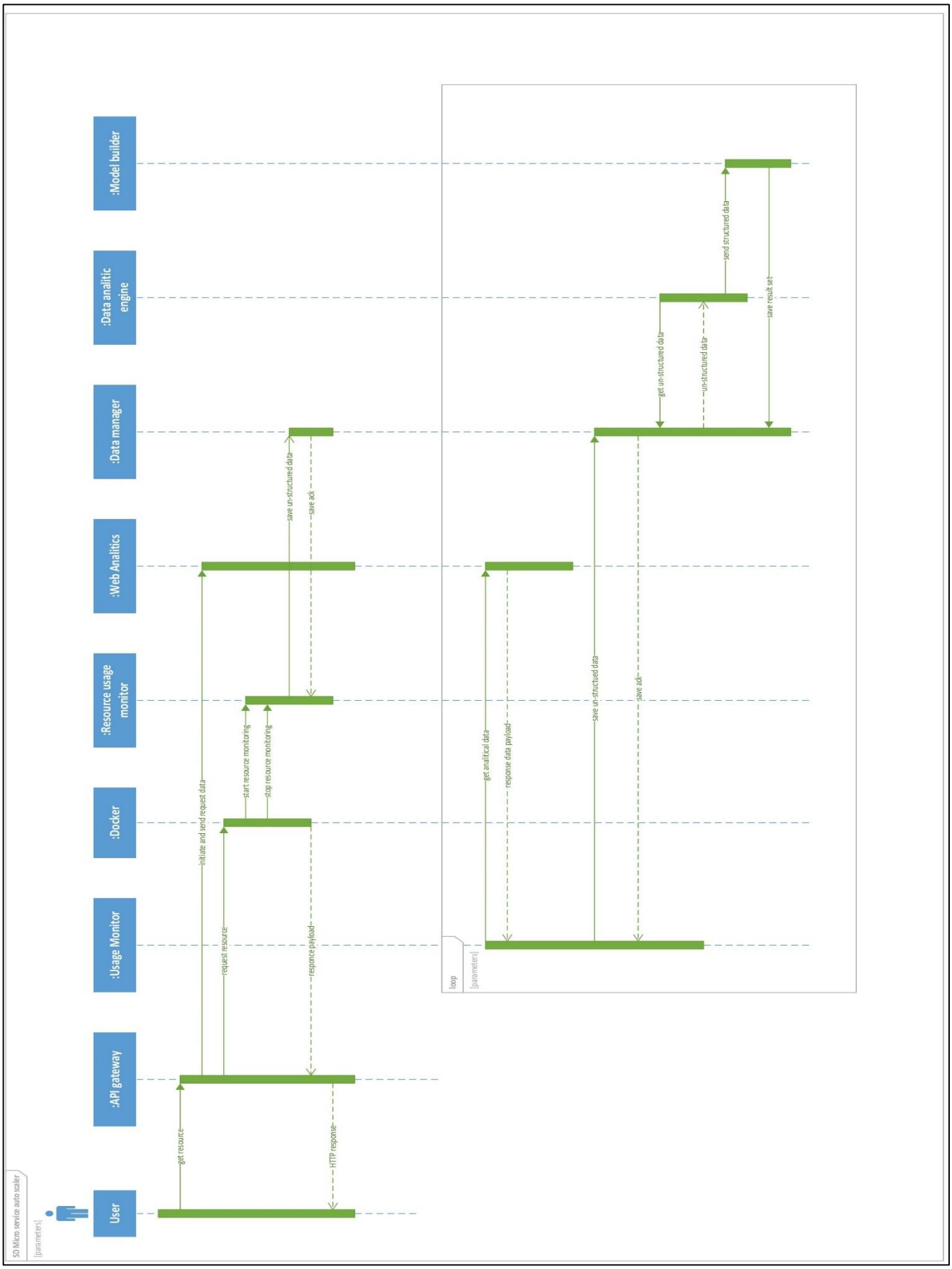

Figure 4: Sequence Diagram

2.6 Assumptions and dependencies

2.6.1 Assumption

- There should be people who are capable of managing the Docker.
- The Docker version should be compatible with the system.
- The micro service for streaming should be up and running.

2.6.2 Dependencies

- Docker should be installed, if necessary the Docker swarm
- The hardware should be sufficient for the system to function.

2.7 Apportioning of Requirements

The section 1.3 of this document provides the overview of the proposed system and the Section 2 provides the detailed overall description on the system and the requirements. The section 3 contains detailed requirements that should be followed while design and implementations. Section 3 describes the preliminary and functional specifications in the first release of the Auto scaler. The methodology of implementing the system may slightly different than the content described in this document during the system design however the requirements specified will not be changed and the systems release will tally with its purpose and objectives.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

User interfaces are used to interact with the users. User interface is that portion of an interactive computer system that communicates with the user. Design of the user interface includes any aspect of the system that is visible to the user. The main UI screens which are available for the system is the main screen, where the deployment takes place and a report view which shows the statistics collected by the systems. This

component will not have any user interface designs. The report generation for this component will be handled by another.

3.1.2 Hardware Interfaces

The system developed will be only require the server for the system to be installed. The server will be needed to be configured, with the open ports, so there will be no conflicts.

3.1.3 Software Interfaces

The system will be developed using python and a Service will need to be developed for accessing certain functionalities of the system, along with services for interacting with the Docker. These services will be developed using the “REST” Protocol, which will be developed using NodeJS.

Software used for documentation formatting to IEEE standards

- MikTex

As far as the software interfaces are concerned, the main interfaces for the resource monitoring component would include the following

- CLI Access Interface:

  This interface will be mainly used for accessing the CLI of the Docker and executing the necessary command. This will be provided as a public interface for the services which have the proper privileges to access and use.
- Data Manager Interface:

  This interface will be available within the micro service, which would provide the formatting of data and the management of the data from multiple sources [Micro Service] to collect the resource allocations and save them for future use. A data manager will be present outside the micro service, which has the functionality of managing structured and unstructured dataset and the predicted result sets.

- Model Building Interface:

  This component will be available outside the Docker system along with the other components necessary for the auto scaler to function. This interface will be primarily used for accessing unstructured data and building a result set from the transformed structured data set.

3.1.4 Communication Interfaces

The main communication mechanism that will be used will the REST implementation that will be made use for getting required data and for executing the necessary commands on the Docker CLI.

3.2 Architectural Design

3.2.1 High Level architecture diagram

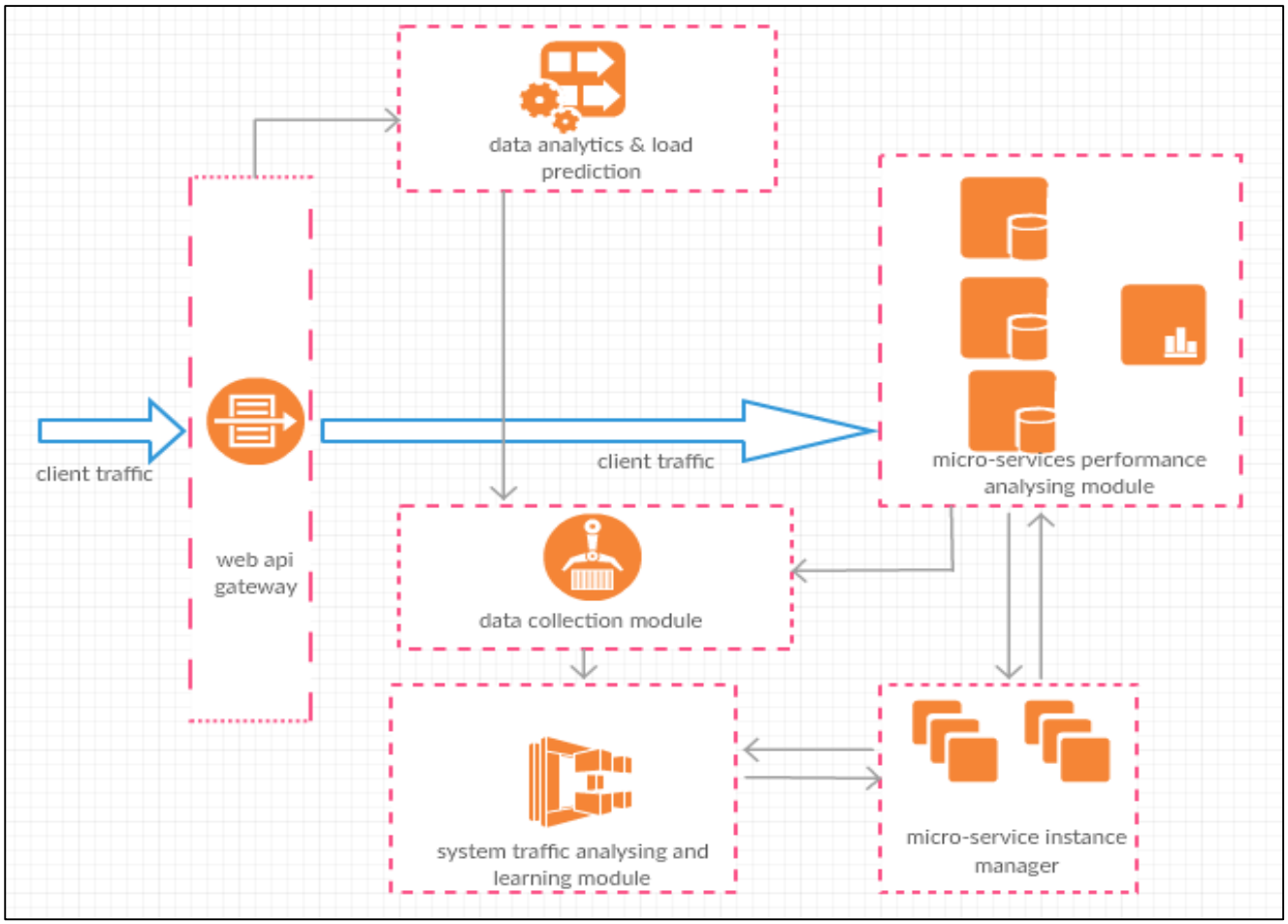

Figure 5: High Level Architecture

3.2.2 Class Diagram

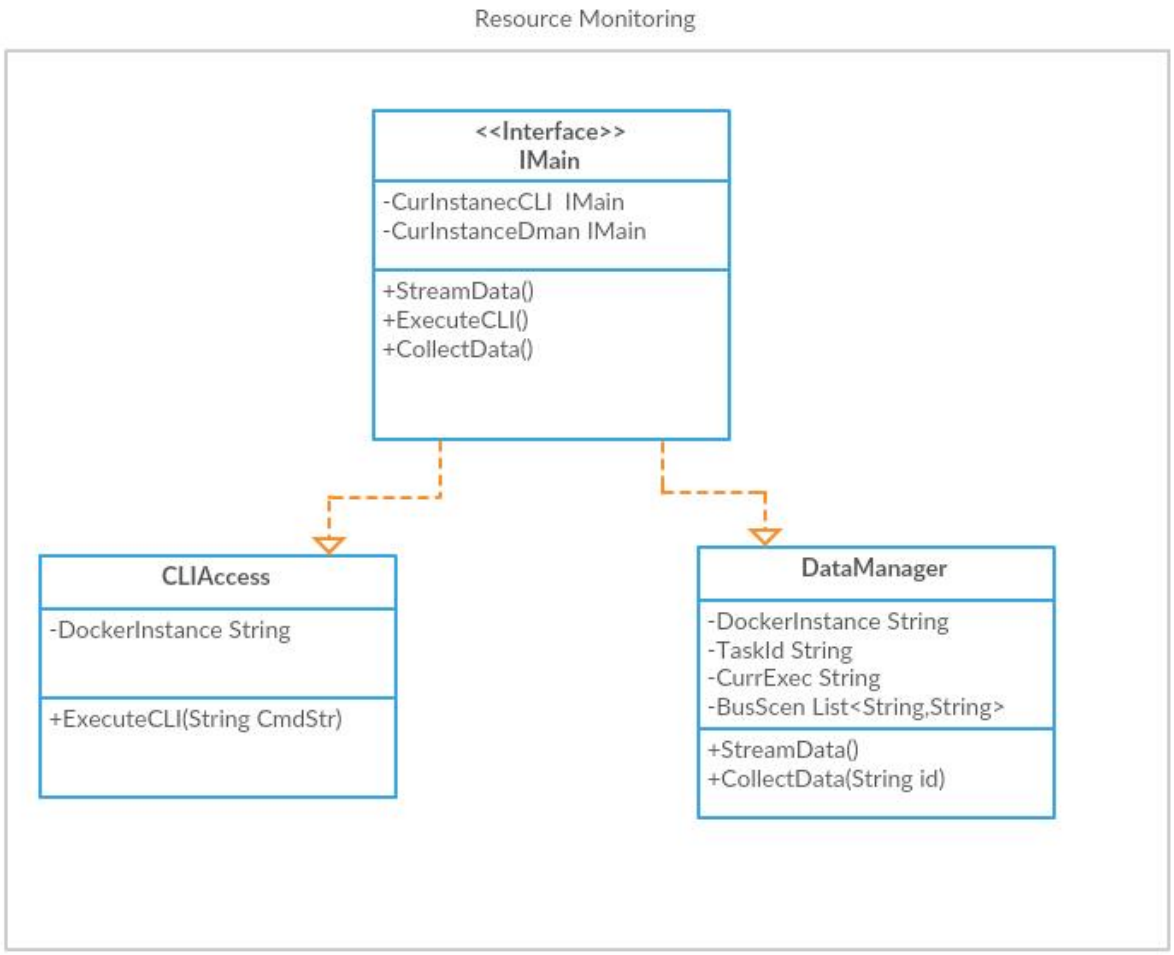

Figure 6: Class Diagram – Resource Monitoring

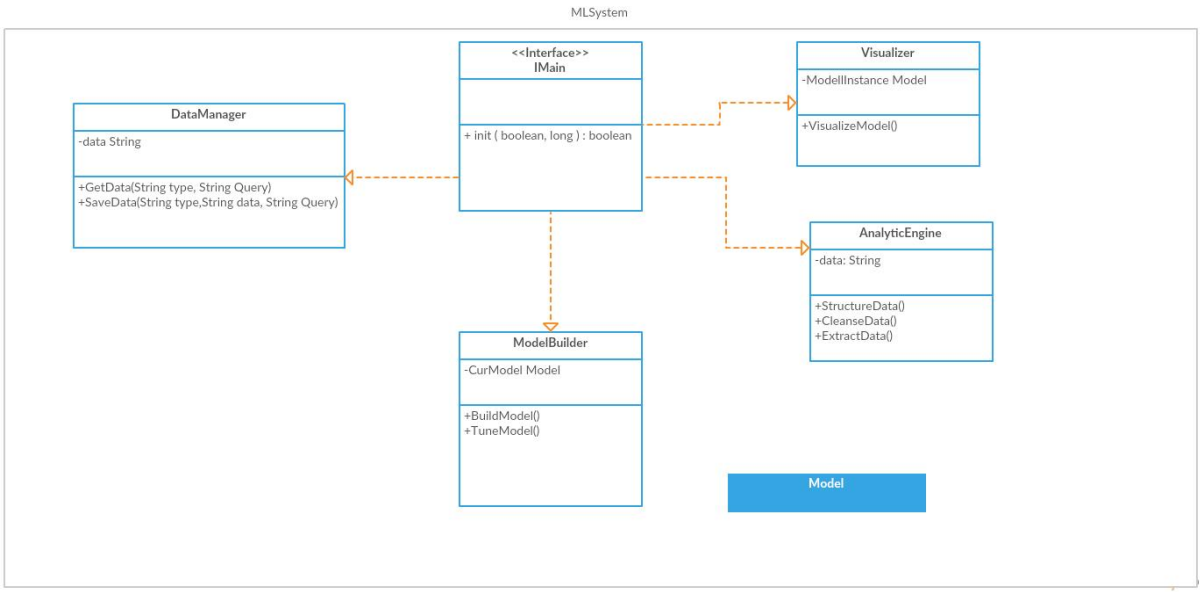

Figure 7: Class Diagram – Machine Learning System

3.2.3 Hardware and Software requirements

2.2.2.1 Hardware requirements

The only hardware which would be involved is the server on which the Docker is deployed. The configuration of this system is the problem in this as the instance manager, as the no of instances require more than the required power therefore it will need to be scalable.

2.2.2.2 Software Requirements

The software requirement for this project is to create interfaces by which it will be possible to communicate between interfaces. This is possible by making use of web services for the communication between each individual components. Other alternatives have been tested during the feasibility, but as the components becomes more highly coupled, it was decided that it would be best to make use of Messaging protocols as an alternative.

3.2.4 Risk Mitigation Plan

The risk for this project includes taking into account the following factors: -

1. Machine Learning being unsupervised.

   As this system will initially be controlled with the administrator's aid, after the system has learnt certain patterns, it will be switched to unsupervised learning mode, thus reducing the need for interaction with the system. This can introduce a risk of the system to get the wrong data thus introducing an anomaly to the set. These offsets will be defined at the initial stage thus reducing the risk of the anomaly being introduced without the necessary parameters being fed to the learning model.

2. Compatibility issue with the installed Docker system.

   The components which will be designed will try to reduce the need for exact versions of the Docker to be installed thus making it more universal. This risk is introduced as our system will need access to the CLI for certain features to function, there is a chance that some features may change over the change in versions. This risk will be managed by making sure that the commands to be used will be available throughout the change history. That is common commands will be used against the pipe operator to achieve the same output.

3. Intercommunication between components.

The components that are used will need to be independent of one another that is they should not affect the other's functionality. This risk is mitigated by making use of web services for inter component communication this also includes, the communication with the Docker from the system.

4. Methods of deploying micro service applications

   The micro service applications can be deployed in 3 main styles which are: -

   - Point to point style
   - API Gateway style
   - Message Broker Style

   The system will need to be interoperable to a level that the method in which the services are deployed will not matter. Therefore, it is necessary to have a layer of abstraction which would make it possible to do so. Thus it is necessary to ensure that the application built by the client and deployed itself will not cause any issues for the developed system.

3.2.5 Cost Benefit Analysis

The product to be built will be an extension of the Docker component that is being currently used as a containerized application deployment and management system. One of the current problem that exist with the said system is the management that is the orchestration and the deployment of the services are time consuming and difficult. It also adds to the cost of maintenance.

The purpose of this system is to reduce the cost when an application is developed using micro service architecture and make it more manageable. This reduces the cost of the system by doing the following

1. Reduce the personnel required for the management of the service.
2. Manage the resources being allocated for the service to function based on its requirement. Proper management in resources enables the reallocation of the resources for other purposes.
3. Have a fail safe way of ensuring that the system is available with the least latency time necessary as possible.

By achieving these, it is possible to reduce the cost that is required for developing, deploying and managing the micro services in a long run.

3.3 Performance Requirements

The main hardware component for the system to function would be the server computer. This system should be able to withstand the increment in the load, until a certain extent at which point more power will need to be added.

3.3.1 Response time

The maximum response time will be considered in milliseconds, with a maximum of 1000 milliseconds and a minimum of 200 milliseconds. The latency of the system should be less to ensure that the deployed systems are properly managed with as low down time as possible.

3.3.2 Processor

The standard processor available on the server machines, are at the moment the Intel I Series, or the AMD Athlon as the core with a maximum of 4.2 GHz. The processing power of the system will need to be configurable based on the requirement change.

3.3.3 RAM

The maximum RAM that are set for a server can be up to 32GB. Whenever necessary the RAM can be incremented according to the requirement of the system.

3.4 Design Constraints

3.4.1 Programming language

The language that will be used to develop the specific micro services will be NodeJS using the Seneca library. The program for developing the Machine learning model that will be used for predictions will be developed using Python. And the REST services that would be required for accessing functionalities, will be developed using NodeJS. If necessary, R will be used for visualization purposes.

3.4.2 Development tools

The tools that will be used for development include: -

- IDE's
    - PyCharm
    - Notepad++

    - Visual Code
- Server
    - Apache OR Apache based server [ex: - Tomcat]
- Docker
- Linux Environment [Preferable RedHat based ex: - CentOS]

3.5 Software System Attributes

3.5.1 Reliability

The main purpose of this application is to create a way for managing the applications developed using micro services, thus it is necessary to ensure that the services being deployed needs to be available with as minimum latency as possible.

3.5.2 Availability

The dependency of this system which is the Docker will need to be available for the system to work. Thus the system will need to be able to access the necessary services whenever necessary.

3.5.3 Security

As this will be an add on to the existing system, it will be necessary to ensure that it does not introduce any security risks to Docker from the developed system. For this purpose, all necessary services will not be exposed fully and will be authenticated before it can access any service of any kind.

3.5.4 Maintainability

The maintainability of the system will be built with the architecture of the system in mind. In the case of this system, it will be built with a plug and use, component based style. Therefore, if the need comes, it is possible to switch the components.

References

[1] RICHARDSON, C. What are microservices? In-text: [6] Your Bibliography: [6]C. Richardson, "What are microservices?", *Microservices.io*, 2017. [Online]. Available: http://microservices.io/. [Accessed: 24- Apr- 2017].

[2] G. Wang and T. Ng, “The Impact of Virtualization on Network Performance of Amazon EC2 Data Center,” in IEEE INFOCOM, San Diego, CA, March 2010, pp. 1–9.

[3] MOHAPATRA, S., SMRUTI REKHA, K. AND MOHANTY, S. A Comparison of Four Popular Heuristics for Load Balancing of Virtual Machines in Cloud ComputingIn-text: (Mohapatra, Smruti Rekha and Mohanty 33-38) Your Bibliography: Mohapatra, Subasish, K. Smruti Rekha, and Subhadarshini Mohanty. "A Comparison Of Four Popular Heuristics For Load Balancing Of Virtual Machines In Cloud Computing". International Journal of Computer Applications 68.6 (2013): 33-38. Web.

[4] FOWLER, M. martinfowler.com In-text: [5] Your Bibliography: [5]M. Fowler, "martinfowler.com", *Martinfowler.com*, 2017. [Online]. Available: https://martinfowler.com. [Accessed: 24- Apr- 2017].

[5] Shridhar G.Domanal and G. Ram Mohana Reddy, “Optimal Load Balancing in Cloud Computing By Efficient Utilization of Virtual Machines” in the Proceeding of the IEEE International Conference on Communication Systems and Networks, Bangalore, Jan. 2014, pp. 1-4.

[6] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In Proc. 6th Annual ACM Symp. Principles of Distributed Computing, pages 1–12. ACM Press, 1987.

[7] MICROSERVICES IN PRACTICE - KEY ARCHITECTURAL CONCEPTS OF AN MSA In-text: [7] Your Bibliography: [7]"Microservices in Practice - Key Architectural Concepts of an MSA", *Wso2.com*, 2017. [Online]. Available: http://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/. [Accessed: 27- Apr - 2017].

[8] “How elastic load balancing works,” [Online]. Available: http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGui de/how-elb-works.html