INTELLIGENT AUTO SCALER FOR MICROSERVICES

Design Document

Project ID: 17-021

Author's Name: D.P Liyanage
Author's ID: IT13148478

…….
Dr. Dharshana Kasthurirathne (Supervisor)

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

May, 2017

Intelligent auto scalar for micro services deployed on the cloud.

DECLARATION

I hereby declare that the system requirement specification document entitled “Intelligent Auto – Scalar for micro services” submitted to the Sri Lanka Institute of Information Technology is a record of original work done by myself under the guidance of Dr. Dharshana Kasthurirathne, Project Supervisor, and this project work have not performed the basis for the award of any Degree or diploma/ associate ship/ fellowship and similar project if any.

Liyanage D.P : …….

Date : …….

Contents

1. Introduction

1.1. Purpose

The purpose of this software design document is to provide a description of the design of the **Intelligent micro service auto scalar** system which focused on auto scaling micro service instances based on the request load which the application is receiving, fully enough to allow the developers of the system to proceed with an understanding on what needs to be built and how it is expected to build. This document provides information necessary to provide description of the details for the software and system to be built.

1.2. Scope

This software design document is for developing a base level system which will work as a proof of concept for delivering base level functionality which can be taken as a feasibility measure for a large scale production system. The main focus of the purposed system is to dynamically scale the micro service instances hosted in **Docker** container management system mainly based on request load which a particular micro service receives.so that the system will not require people to handle the scaling of the application. Load anticipation is also a main functionality of the system where it will analyze the request load patterns and anticipate for a predicted load that may happen in the near future.

This following components will be covered throw out this software design document

- Usage pattern analysis
- Resource usage monitoring and execution plan generation
- Decision making the micro service instance scaling
- Business scenario prediction

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

1.3.1.1. Client micro services application

The application which will be designed and implemented by the clients of the purposed intelligent micro service scalar.

1.3.2. Acronyms

- DNS[1]
    - Domain name service

1.3.3. 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

1.4. References

[1] DOMAIN NAME SYSTEM **In-text:** [1]

Your Bibliography: [1]"Domain Name System", *En.wikipedia.org*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Domain_Name_System. [Accessed: 01- May- 2017].

1.5. Overview

This document is sectioned into 2 main categories where each category may have number of sub categories. First section of the document will state about the overall architectural design of the full application while later section will mainly focus about the architecture and the design of the Micro service instance manager component.

2. Overall Description

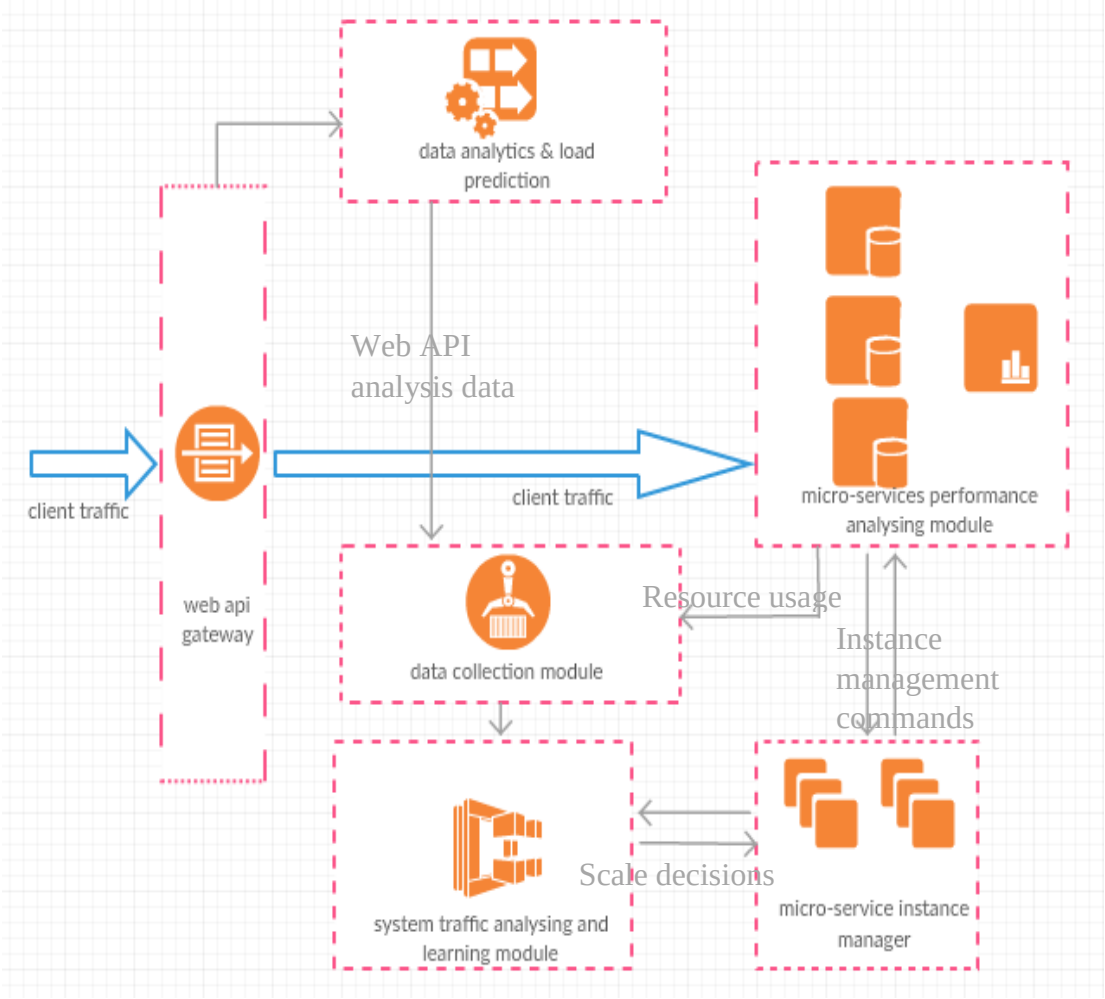
This section will provide an overall idea about the full system architecture and design. It will consist of all the components which will be developed by the other members of the group and will contain a brief architectural introduction to those components.

2.1. Product perspective

The main focus of the system is to scale the micro service instances hosted inside a platform known as Docker. The purposed system will be an external supportive extension for the Docker platform. Most of the time micro services as deployed as containers into Docker platform. Container is a self-contained application implementation which can be executed on top of Docker platform as well as it protects the independence of the implemented code from other micro services implementation. This advantages have made the Docker container the world slandered way to bundle the micro services applications.

The system will consist of four components, which will handle the request load analysis, resource usage analysis, micro service scaling manager and a business scenario visualization module. Where these modules will collect the required information and analyses them to so that the system can make the required decision making which will be then used for the application scaling or de-scaling. A micro service which will be deployed to the Docker platform will going to be act as the client for the Docker where it will expose an interface which can be used by the systems modules for data collection and micro service scaling. Further explanation about the components and the responsibilities will be provided below.

As the system heavily relies on big data analysis, a dedicated storage will be there inside the system. The purpose database system will be a no-sql data base. The application setup procedures will be discussed in the later sections. Given below is an overall system architecture which indicates how the explained components interact with each other and with the Docker platform.



2.1.1. System interfaces

The system will be containing one system interface with a Web request analytics server. The interface should be able to communicate with the provided analytics servers like Google web analytics, WSO2 web data analytics etc. the interface will reside inside the Web API analysis component where it can be used to get the analytical data to make the required predictions about the load patterns.

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

- Linux kernel version 3.10 or higher
- CS Docker Engine version 1.13.0 or higher
- 4.00 GB of RAM
- 3.00 GB of available disk space
- A static IP address

2.1.3. Software Interfaces

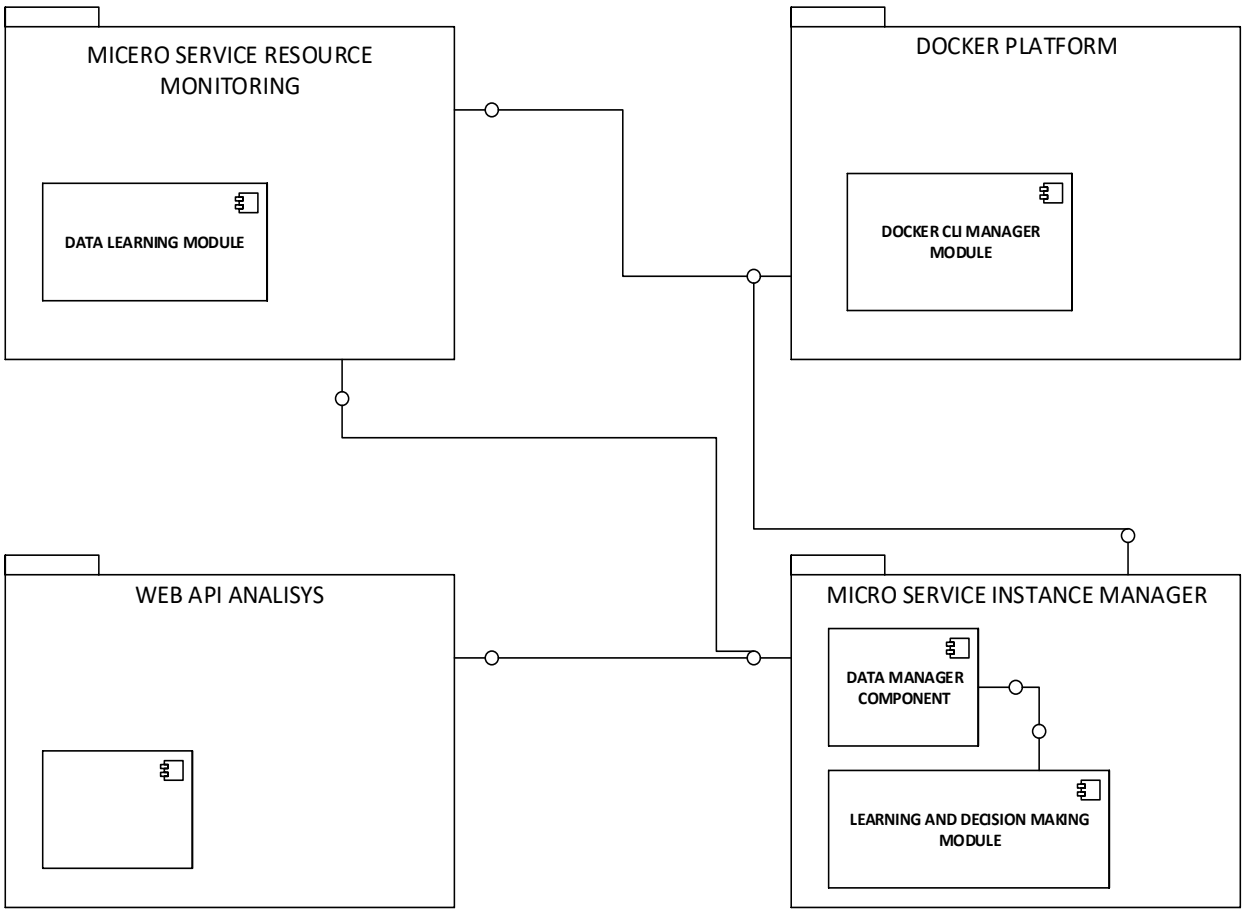
As the proposed system is an extension for the existing system, the software interface will be the full installation of the Docker system on to minimal required system requirements. While this is also part the following ports will be needed to be kept open or the configuration should be changed for the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

The following table provides the necessary information for the ports which are needed to open for Docker platform to continue.

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

Furthermore, given below is the system high level system component diagram which shows the interconnection between the components and the software interfaces.



2.1.3.1. Micro service resource monitoring

This module will contain two software interfaces

1. A software interface for the interconnection with the Docker CLI manager module inside the Docker platform which will be used to receive the micro service execution information and the resource information required by the micro service resource monitoring module.
2. A software interface will be available for the communication with the micro service instance management module, which will be used to stream the analyzed data.

2.1.3.2. Micro service instance manager

Micro service instance manager will contain three software interfaces

1. A software interface will the available for receiving data from Micro service resource monitoring module and the web API analysis module.
2. An interface to give micro service scale management commands to the Docker CLI manager module
3. Inside the component two private software interfaces will be available

2.1.3.3. Web API analysis component

This component will contain one software interface to communicate with the Micro service instance manager component. The interface will be used to send the analyzed and predicted data to target component so that the data can be used for the instance management decision making.

2.1.3.4. Docker CLI manager component

This component will reside inside the Docker platform and will contain one interface which will be used to stream the generated data to the micro service instance scale manager module

2.1.4. Communication Interfaces

Communication between the system components is a main consideration of the purposed system because the system functionality is fully dependent on data which are produced on each component. Other than the system inter-module communication the purposed system needs to communicate with the Docker platform as well. The Docker CLI manager component abstracts the communication by managing the Docker CLI inside the component.

2.1.5. Memory Constraints

System needs to have the enough memory capacity to run the Docker platform and the purposed system. The minimum requirements are specified above.

2.1.6. Operations

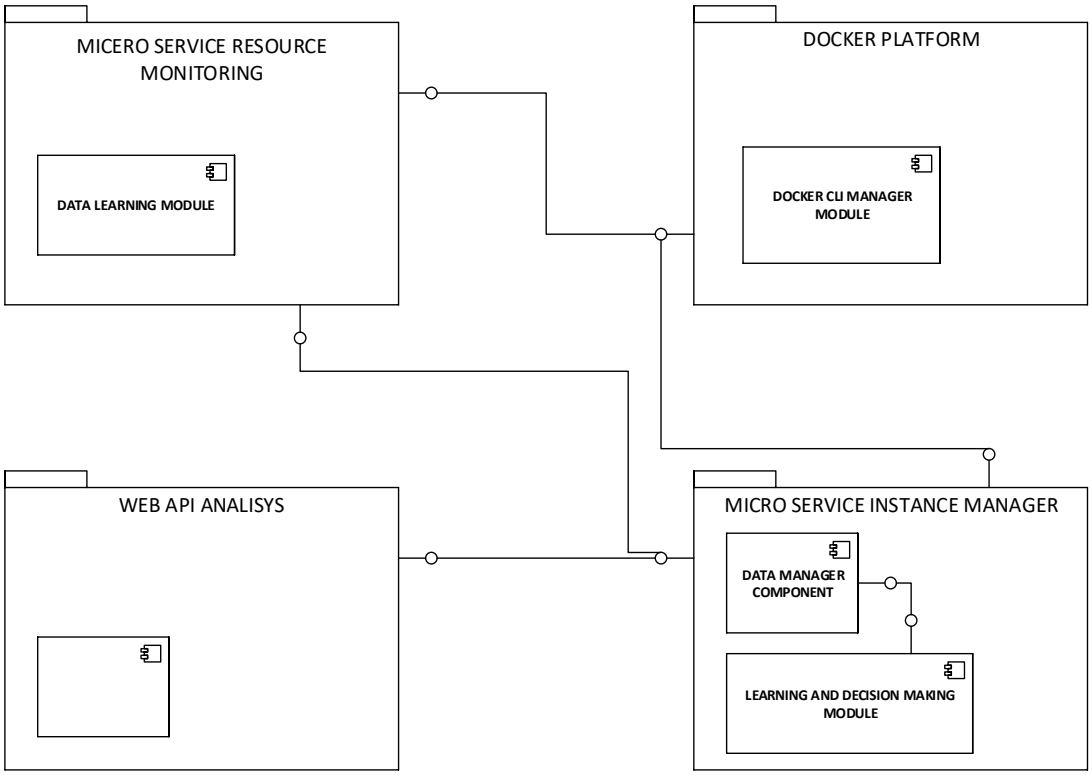
- System needs to be installed on a server and the system administrator must provide the required permissions and the setup requirements accordingly.
- Docker CLI manager module should be installed inside the Docker platform.
- Client system administrator should configure the web API DNS with the Micro service scalar system.
- System administrator should open the required ports in the servers as specified at the top of the software design documentation.

2.1.7. Site adaptation requirements

The system will be deployed on top of Docker platform, so the system will not need any site adaptation procedures as Docker containers are platform independent.

2.2. Product functions

This section will provide information about the main functions and the components of the proposed system. System will contain four main components.



2.2.1. Web API analysis component

Will serve as the starting point of the system. The main functionality of this component is to analyze the web API gateway to gather data on request load for a scenario, load patterns, request resolution time etc. which will be used as a parameter in the micro service instance scale management module to make the required decisions.

2.2.2. Micro service resource monitoring component

This is the component which is responsible for analyzing the resource usage each micro service which will be used as a parameter in the micro service instance scaling management component. Apart from that the analyzed data will be used to identify the anomalies of micro services like memory leaks, development issues etc.

2.2.3. Micro service instance manager component

The core functionality of the purposed system is to scale the client micro services system up or down based on the load patterns, and related information. Micro service instance manager is the component which is responsible for handling this requirement. Component should get the required data from the above mentioned components and make necessary decisions to manage the number of instances which should be running on the Docker platform at a selected moment.

This instance scaling functionality is carried out with the combination of the Docker CLI manager which is hosted inside the Docker platform itself.so the manager component will send the required commands to the Docker CLI manager component.

2.2.4. Business scenarios visualization module

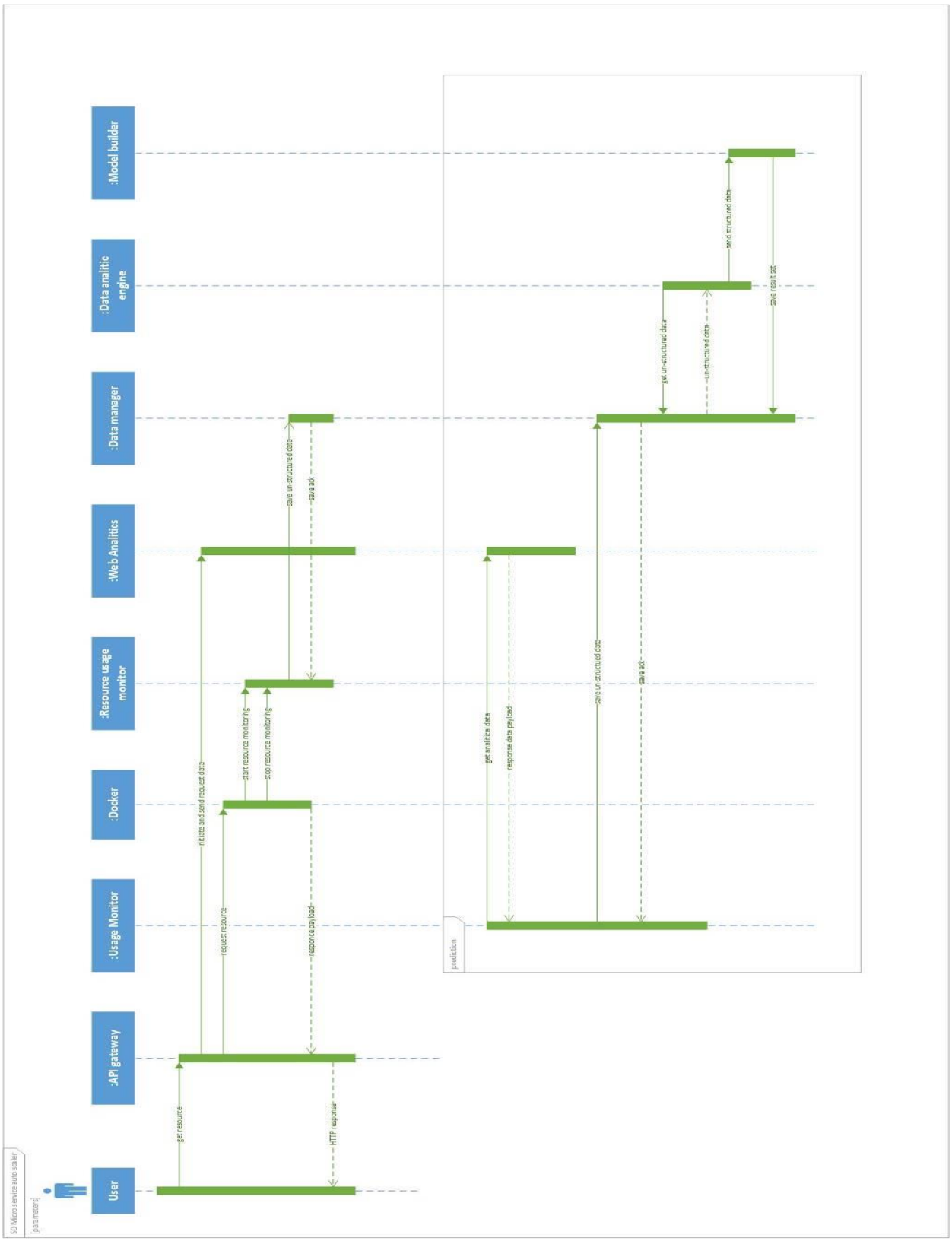
This component is responsible for visualizing the business scenarios which are being executed inside the client micro service application. Business scenario visualizing module in intended to get the required information from the micro service analyzing module. The micro service execution pattern will the streamed by the micro service resource analyzing module which will be subscribed by the business scenario visualization module combined with the

request URLs so that the module can generate a relationship between request URL and the executed micro services which will then be used for the visualization of the business patterns reside in the system.

2.3. User characteristics

The auto scalar 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. System high level sequence


SD Micro service auto scaler
[parameters]
User
:API gateway
:Usage Monitor
:Docker
:Resource usage monitor
:Web Analitics
:Data manager
:Data analitic engine
:Model builder
get resource
HTTP response
initiate and send request data
request resource
responce payload
start resource monitoring
stop resource monitoring
save un-structured data
save ack
prediction
get analitical data
response data payload
save un-structured data
save ack
get un-structured data
un-structured data
send structured data
save result set

2.4.1. User

User can be any real world person who uses client micro services application. The user will initiates a request for the client application which will be received by the web API gateway.

2.4.2. Web API gateway

Web API gateway also will be implemented by the clients of the Intelligent micro service auto scalar, which will abstract the underline micro services. The request made by the client will get received by the web API gateway and the request will be converted to a format which can be identified by the client micro services. Web API gate way will be bounded to a Web analytics system by the scalar system. So when a request is received by the web API gateway it will be asynchronously propagated to the web analytics server for analyzing the request and the request-response resolution time etc.

At a successful processing of request, the underline system will send a response payload to the web API gateway and it will be returned as a HTTP response to the client.

2.4.3. Docker

Docker is the container manager platform used in the application which will manage the life cycle of each micro service running. The public interface of each container will be exposed to as the configuration of the application. The exposed interfaces will receive the request which is got forward by the web API gateway.

When a request is received by a particular micro service, the respective micro service will direct a message to the micro service resource analysis module which is also hosted inside the Docker platform. The message will be sent on the start of the execution and on the execution of the micro service.

At the end of the execution of the request an endpoint micro service will return a result payload to the web API gateway.

2.4.4. Usage monitor

Usage monitor component will request data from the web analytics server to get the load analytics and the request-response resolution time. And send the received data to the Data store module after the required document.

2.4.5. Resource usage monitor

This module will receive the asynchronous messages which will get initiated by the micro services inside Docker platform. This module will gather the information required and at the stop execution stop message from the micro service instance it will sync the gathered information to the Data manger module.

2.4.6. Data Manager

Data manager module will gather the data which will get received by the Resource usage monitor module, and the usage monitor module. It will send an acknowledgement to the resource usage monitor module.

2.5. Constraints

The client micro service application should implement a Web API gateway to abstract the underlying micro services.

2.6. Assumptions and dependencies

This section will describe the dependencies and the assumptions which are made when designing the document.

2.6.1. Assumptions

- 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.

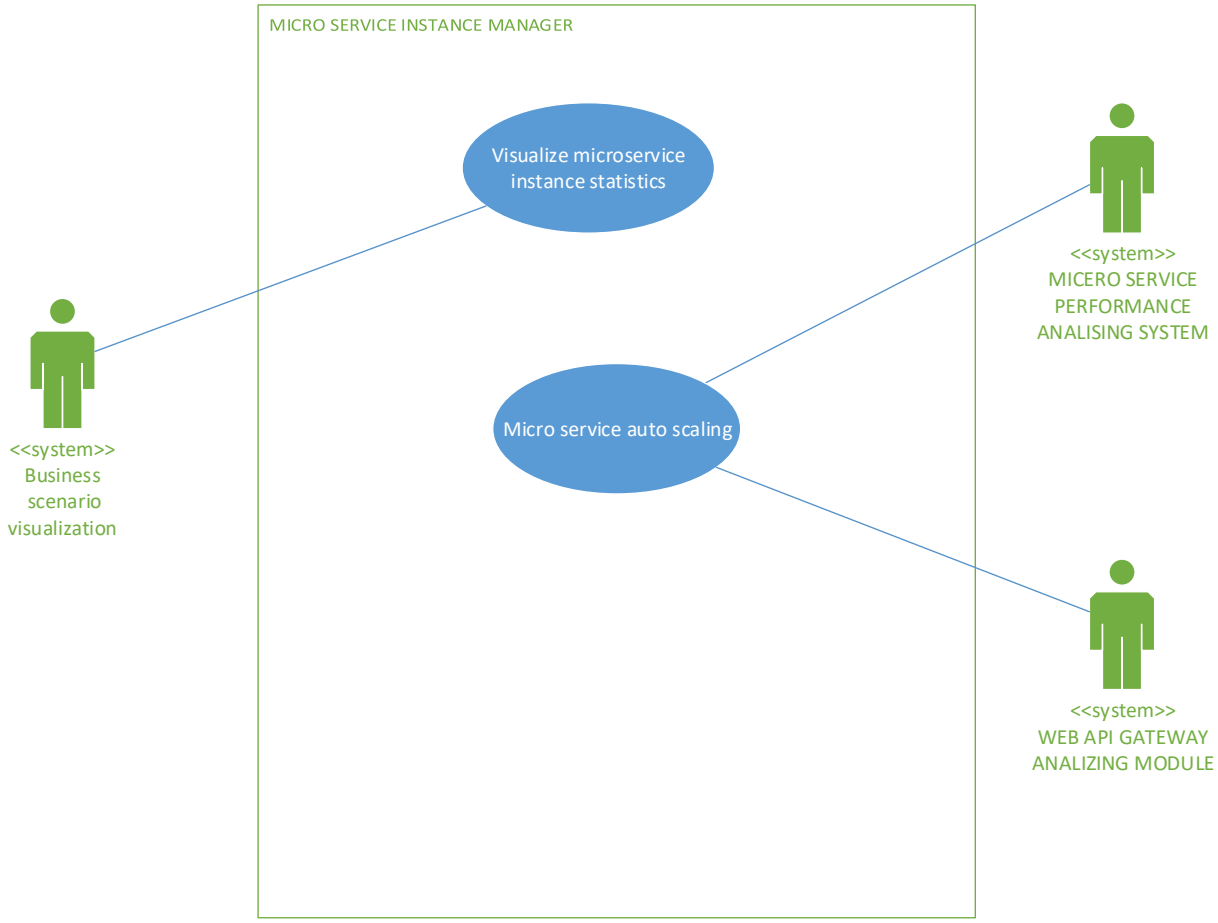
2.7. Apportioning of requirements

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 scalar. 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

In this section the main focus is given to the micro service instance manager component which will be responsible scaling the micro services up or down. This section will covers the design of the described component.

3.1. Use cases



This component will not be accessed by external human clients but will be invoked by the internal components. So the system actors would be the internal sub systems.

3.1.1. System actors

3.2. External interface requirements

3.2.1. User interfaces

Micro service instance scaling component is a private system internal system component which will not provide external user interfaces. But the component provides required statistical data which will be displayed to the user in Business scenario component

3.2.2. Hardware interfaces

The purposed module will not contain a direct hardware interconnection. So the module do not need an explicit hardware interface

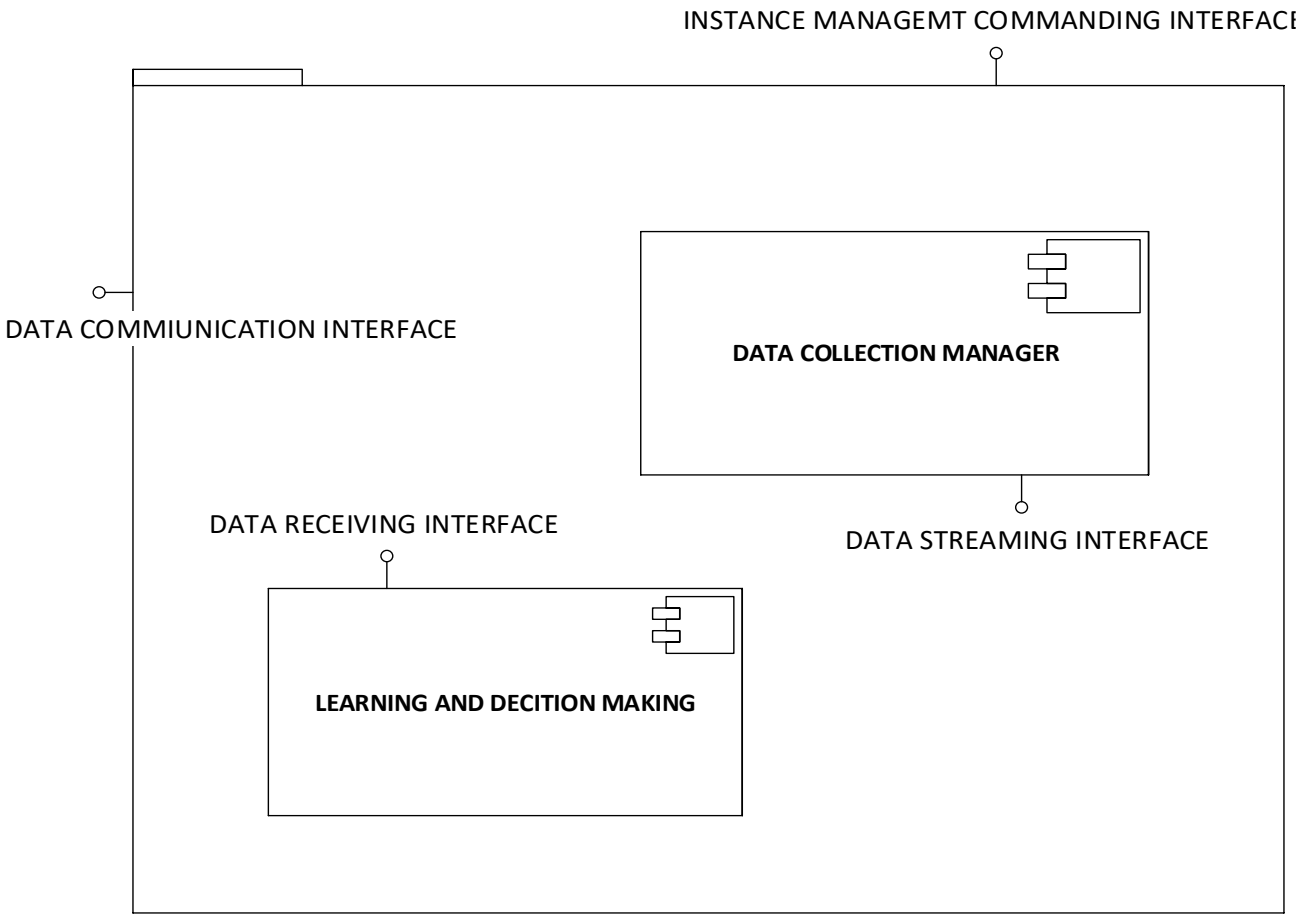
3.2.3. Software interfaces

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 Micro service instance balancing component would include the following



Micro service instance manager contains four interfaces which are intend use on inter-component and inter-module communication.

3.2.3.1. Data communication interface

This is the interface which will be used to receive required data from the Web API analysis component and the Micro service resource analysis module which will be used inside the component for the analysis and the decision making procedures.

3.2.3.2. Instance management commanding interface

This is the interface which will be used to issue the required commands to the Docker CLI manager module which is hosted inside the Docker platform where it would convert into a Docker CLI command and get executed inside the Docker platform for the purpose of scaling the instances up or down.

3.2.3.3. Data receiving interface

This is a private interface which is intended to control the input format of the data stream which is required by the Leering and decision making module.

3.2.3.4. Data Streaming interface

This is a private interface inside the Micro service instance manager component. This interface is used to send the collected data to the **Learning and Decision making module** which is also reside inside the Micro service instance manager component. The motivation for this interface is the main data receiving interface will receive two types of data from the Wed s API analysis component and the Micro service resource analysis component which need to be formalized to a common format so that it can be used inside the Learning and Decision making module.so this interface will stream a formalized set of data to the client module.

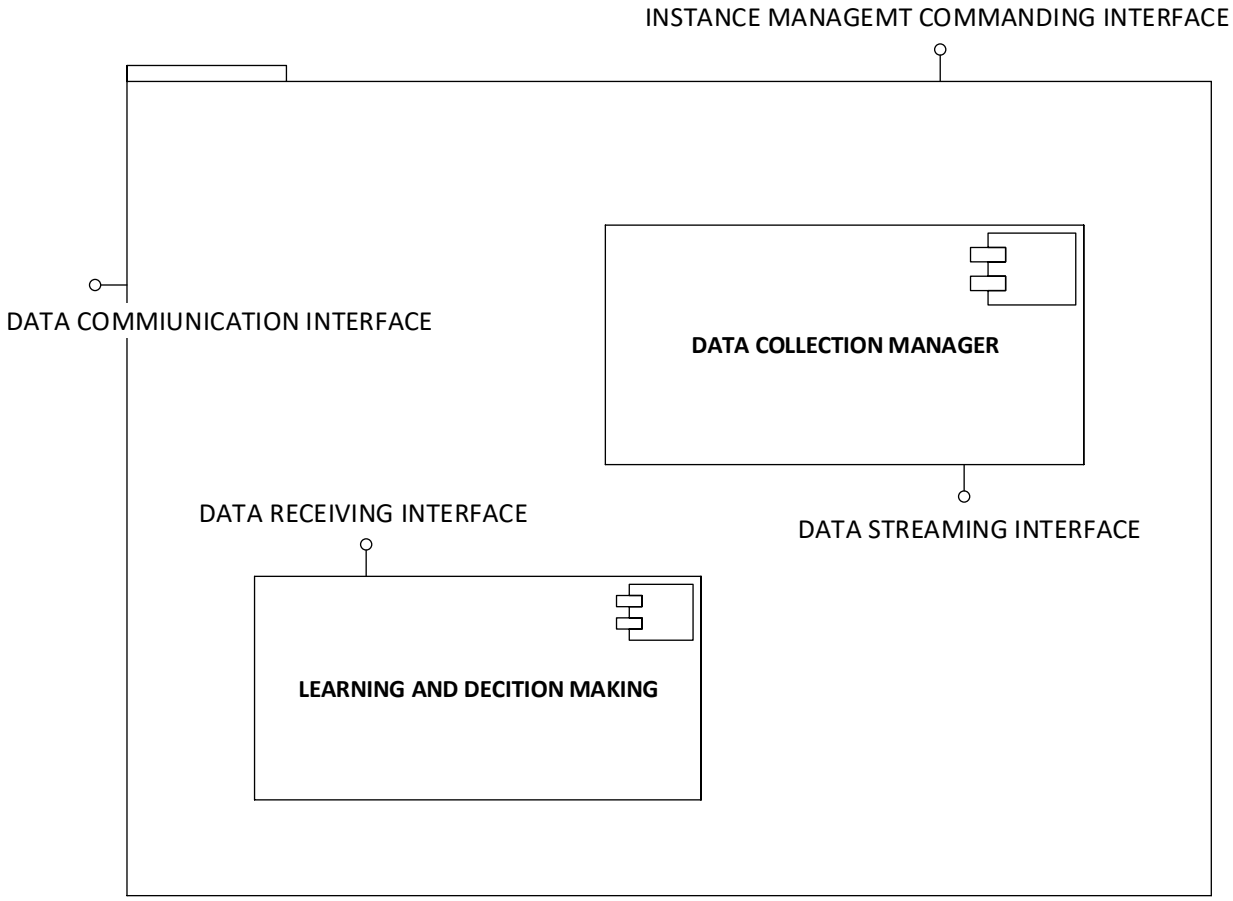
3.2.4. Communication interfaces

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.

3.3. Architectural design

In this section purposed architecture for the Micro service instance manager component will described in detail.

3.3.1. High level architecture diagram



The main purpose of this component is to make the required decisions to scale the running micro services for a particular micro service up or down and make the required anticipation decisions based on the combination of request load patterns and the micro server resource usage patterns. This component will contain two main modules, which are

3.3.1.1. Data collection manager component

Main purpose of this component is to collect the data received by the external components and covert it in to a canonical format where it can be used inside the Learning and decision making module. There are two types of data that the module is received from Web API analysis module and the Micro service resource analysis module. The component will collect the data and covert to the canonical format and stream it to the Learning and decision making module through the Data streaming interface. Canonical format will not be described here as the format is subjected to changes in the near future.

3.3.1.2. Learning and decision making module

Learning and decision making module is the main module of this component and responsible for making the required decisions which are streamed to the Docker CLI management module which is hosted inside the Docker platform. The decisions are carried out using a machine learning module where it analyzing the data received from the Data collection module.

The data will contain the request load patterns and the resource usage patterns of the running micro service instances and the web API gateway where the system use machine learning the model the relation between the API load pattern and the request load pattern of the microservices.so the module will identify the increase of a load and the thresholds of the implemented micro services.so the decision is made to increase the number of the micro service instances that are running on the Docker platform or to decrease it. The decisions will be transferred to the Docker CLI manager. The purpose of the Docker CLI manager will be described in the later sections of the document as the module will perform more of general purpose actions which will be used by the Micro service resource manager as well.

And the other purpose of this component is to make anticipation decisions based on the data which the components will receive. Component will use machine learning to analyses the load patterns and extract regular patterns from the data which is collected over time and predict the future load and make the required scaling based on it.

3.3.1.3. Docker CLI manager component

This component is a multipurpose component. It mainly forced on abstracting the Docker CLI where the Docker CLI abstraction is used for many functions like getting resource usage of the micro services and the executing the CLI commands on Docker platform. Micro service instance manager component will be using the CLI commands execution facility of this component in order to scale the running micro services up or down. So the purposed component will convert the received command to a corresponding Docker CLI command and do the execution. The generated result would be transferred back to the component which made the request. This component will be a micro service itself which can be scaled based on the requirement of the core application.

3.3.2. Software and hardware requirements with justification

Micro service instance manager component will not state any specific hardware or software requirements.

3.3.3. Risk mitigation plan with alternative solution identification

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.

3.3.4. 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.

3.4. Performance requirements

Since the purposed system works as a middleware for the client application, it should not impact the performance of the client application or the impact should be minimum as possible. Specific performance requirements will be listed down below

3.4.1. 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.

3.5. Design constraints

- The client system should be implemented on top of Docker platform.
- The client system should have a web API abstraction over the application deployment.

3.5.1. Development tools

The tools that will be used for development include: -

- IDE's
    - PyCharm
    - Notepad++
- Server
    - Apache OR Apache based server [ex: - Tomcat]
- Docker
- Linux Environment [Preferable RedHat based ex: - CentOS]

3.6. Software system attributes

3.6.1. Reliability

The system should be highly reliable in scaling up or down as the full system performance depends the robust scaling of the micro services.

3.6.2. Availability

The purposed system should have a high availability as the load can vary instantly in any time. So that the application can respond to the load changes as required.