INTELLIGENT AUTO SCALER FOR MICROSERVICES

17-021

Project Proposal Report

Dinesh Priyankara Liyanage

Aleem Mohamed Firnas

Sinthujan Ganeshalingam

W.J Abhayarathne

Bachelor of Science Special (honors) in Information Technology

Department of Information Technology

Sri Lanka Institute of information technology

March 2017

DECLARATION

We declare that this is our own work and this project proposal does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning 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.

Name	Student ID	Signature
Dinesh Priyankara Liyanage	IT13148478	
Aleem Mohamed Firnas	IT14064432	
Ganeshalingam Sinthujan	IT14056826	
W.J Abhayarathne	IT10230848	

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

……. ……..

Signature of the supervisor: Date

(Dr. Dharshana Kasthurirathne)

ABSTRACT

The SOA [Service Oriented Architecture] which was introduced in the early stages of web communication has come a long way to the point that most applications cannot exist without some form of implementation with this architectural style. Many variations of the said architecture have emerged over the years where some have particular advantages over the base while others have some disadvantages by design.

One such architecture is the *Micro Service Architecture.* This architecture has been called many things amongst which it is called a “fine grained SOA”. The architecture mainly focused improving productivity through maximizing the ability to do independent development and deployments. So, that the developers or in fact development teams can work independently from other development teams and deliver the product when they are ready. All modules are developed independent of one another therefore the modularity of the system is increased while coupling between each module is decreased. Applications developed with this architecture are easy to maintain as multiple instances of the same service can be initialized for rerouting traffic while the modules are updated [1]. From the business point of view this is a huge advantage compared to monolithic architectures as micro services architecture addresses the rapid requirement and business process changes. But on the other hand it introduces a lots of new complexity to the software development life cycle and it requires the developers to fulfill a certain level of maturity to work on a micro service application and the deployments and monitoring processes should be automated. Most of the companies with less human resources and automation capacity do not go for micro service architecture because of this reasons.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1.Background

Most of the enterprise applications built on SOA seems to be running well in past few years, there are so many challenges facing in this available architectural solution. The monolith applications are about building the components and packaged as a single solution and deployed. The drawbacks of this style are scalability, availability of the non-breakable components, dependable work for the developers, increase of the development time etc. The Micro services addresses these issues and can be used to restore the flexibility that may have been lost in SOA.

Micro services is a way of breaking larger software projects into smaller, independent, and loosely coupled modules. Individual modules are responsible for defined and discrete tasks and communicate with other modules through simple, universally accessible APIs

One of the major consideration of any kind of architectural design is scalability. By the Canonical definition given by *Martin Fowler* [2], 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."*

Micro services are normally deployed into a virtualized Container management system of which an example is the *"Docker"*. By normal standards, a single micro service will be deployed as a single Docker container but it is possible to run multiple micro services in a single container as well. The problem which we have identified with the existing system is that it allows scalability but this requires

more human resources as well careful planning and management which is the main reason that many companies do not go for this architecture even with its advantages. So even if they initially begin by developing a micro service architectural application, the product would end up as a monolith application because they do not possess the required technical skills and human resources to manage the application.

The main or core difference of the micro service architecture with the SOA architecture is the modularity of the system. The modularity increases the advantage of developing and maintaining the system while at the same time it also increases its complexity. For example if we were to take a SOA system which has exactly 50 cohesive services and this was to be converted to a system using the micro service architecture, the end result of the project would make it difficult to orchestrate the calls in between each micro services and the calls between the system and the gateway. Therefore it becomes necessary to have teams to maintain said services. This is a problem present in the SOA architecture as well but the amount of resources

Needed for the management of a system built on top of the micro service architecture is high compared to a system which is monolithic.

Micro services can be deployed in 3 different manners:-

1. Point - to - point style

In this style, every services in the system is considered an endpoint and the services can communicate directly with one another. This works fine for services which are relatively simple but as the system becomes more complex it becomes difficult to keep track of it. The architecture is given below

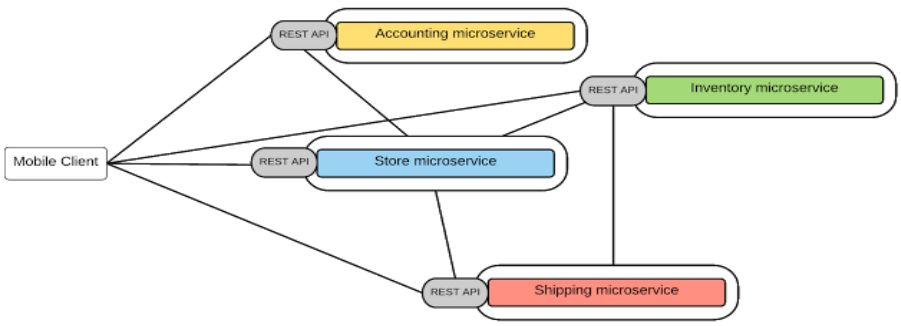

Figure 1: Point-to-point style [14]

2. API Gateway Style

In this style the idea is to have a centralized gateway which will act as the endpoint for all requests. This avoids the hassle of keeping track of the endpoint as with the point to point style. The flow is described in the diagram below

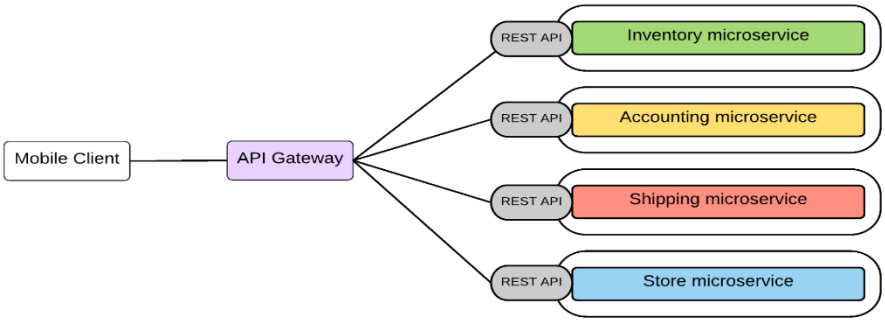

Figure 2: API Gateway Style [14]

3. Message Broker Style

In the message broker style, the services are deployed in accordance of the event based architecture, in which the subscribers and publishers are present and the services are executed when an event is triggered

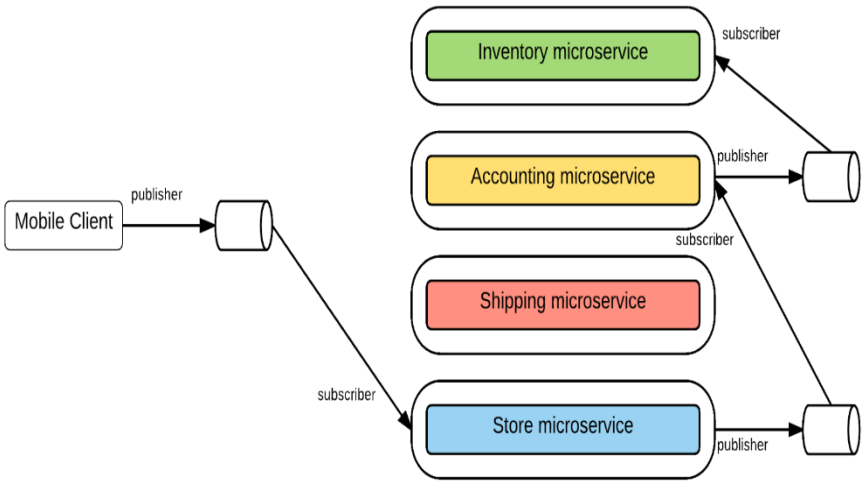

Figure 3: Message Broker Style [14]

The advantages of using a micro service architecture over the traditional SOA are:-

1. Allows each individual components to have their own lifecycles.
2. Makes it easier to scale as necessary.
3. Breaks the technological dependencies between components.
4. Makes it easier to analyze each component.

The advantages mentioned above if further explained are, that each individual components can have its own way of deployment, based on the components individual requirements, it is not bound to the boundaries defined by the system as a whole. The 2nd point is that it is possible to scale only the required service rather than to scale for the system as a whole thus reducing the amount of computational resources and reducing the cost. The 3rd point is that it is possible to implement each individual service in different languages as per the requirement of the component and communication is possible between them. Therefore the system as a whole does not need to limited to one language's benefits but can take maximum usage of other languages as well. The last is the most important advantage of the micro service architecture, i.e. the components become easier to analyze individually. If the system is properly following the standards defined by the micro service architecture then it would be possible to make changes to components, replace or remove them whenever necessary without impacting the system as a whole.

The current system needs heavy human interaction to manage each of these Docker containers to achieve the expected performance of the system, the proposed system will be to implement a load balancer which would be able to automatically scale the containers while not requiring any human interaction at all and distribute the load among multiple running instances of the containers based on the analysis performed from the data gathered using our system [3].

Developers tend not to use this architecture as they find it difficult to maintain the architecture as they continue adding functionalities, in the end the application becomes a monolith SOA architecture based application even if it was initially built on top of the micro service architecture. The reasons for this can be either

1. Poor management skills on maintaining the application.
2. Not enough resources to manage the application.
3. Or there are not enough people with the expertise to maintain the application.

Our final outcome of this project to create an automatic scalar which will dynamically manage the containers based on the load thus saving computational power which could be redirected for other purposes. This also will include the reduction of the resources required for the management of the deployed system. The proposed solution will try to take full advantage of the architecture as much as possible.

1.2.Literature Survey

With the introduction of the micro services architecture researchers came to the understanding that this is the most anticipated architecture for the applications which followed service orientation. But with the advantages it brought complexity to the management of the application.so researchers started to find ways to eliminate this disadvantages from the architecture because they have identified micro services architecture is worth finding solutions for its disadvantages than throwing away and use the monoliths again. so that, companies like Google also initiated projects to solve this matter, thus there are some open source products to manage micro service applications, but still the comparatively smaller companies would not go for a micro services implementation because the available products are not in mature enough to be used on a wide scale.

Cloud computing architectures are inherently parallel, distributed and serve the needs of multiple clients in different scenarios [4]. This distributed architecture deploys resources distributive to deliver services efficiently to users in different geographical channels. Clients in a distributed environment generate request randomly in any processor. So the major drawback of this randomness is associated with task assignment. The unequal task assignment to the processor creates imbalance i.e., some of the processors are overloaded and some of them are under loaded. The objective of load balancing is to transfer the load from overloaded process to under loaded process transparently. A research has been done on comparison of four popular load balancing algorithms of virtual machine in cloud computing, have discussed starting with dividing the types of algorithms as static and dynamic.

static load balancing algorithms - is easily carried into execution and takes less time,

which doesn't refer to the states of the load nodes, but it can be only used in certain specific conditions.

1. **Round-Robin Scheduling Algorithm**

Round-Robin Is only applicable to the circumstances in which all the nodes in cluster have the same processing ability, and it is the simplest one which could most easily be carried out. [5]

2. **Weighted Round-Robin Scheduling Algorithm**

Is designed to better handles servers with different processing capacities. Each server can be assigned a weight, an integer value that indicates the processing capacity.

3. **SWIM**

This is a protocol that was developed for checking the live status of the micro services individually. This is used for large scale process groups that is a collection of micro services. Each neighboring micro services will be receiving a heartbeat from the other to ensure that the live status of the micro service. This was developed because of scalability issues, of the traditional heartbeat protocols which strained the network as the no of micro services deployed increases with the increased network load, it compromises the response time of the application. That is it consumes the computational resources of the server. [6]

4. **Docker**

In the research conducted by David Jaramillo, Duy V Nguyen and Robert Smart from Cloud Engineering Services [7], IBM has discussed the concepts in Micro services. They have mentioned the drawbacks of the monolith applications where almost all of the application developers or stakeholders having the considerable amount of attention and described about the areas which needed to be consider most when developing an application for any architectural design. They have identified “Micro services” architectural pattern as a solution to almost every mentioned issues. They have presented in a way of how micro services is a most beneficial solution compare with traditional monolith application where highly

coupled component's deployment and hardness of the scalability etc.

Since the Docker has been introduced they did analyze about the each component in Docker and explained how the Docker can help to build the applications and utilize the micro services' core concepts etc. Even though Docker itself is not a silver bullet which can address every challenges of building a micro service architecture.

1.3.Research Gap

The main requirement of this research is to create an intelligent micro service scalar which will learn when to scale up and down, what are the most important business scenarios which needs the priority without a user interaction.at the same time it is intended to create a graphical user interface where it would display the business scenarios and which micro services are aggregated to complete a specific scenario as there are the most crucial limitations that are there in the available products.

The major focus of the intended system is to create an intelligent micro service scaling system which will do the required scaling without getting user inputs. Since existing system do not provide the functionality to do the scaling process without user interaction. This is a critical function for micro service eco system because most of the small companies do not go for micro services architecture as they do not compose the required skilled human resource to the management of the application.so if the application can grow and sink on its own without the interaction of humans, the company needs only to develop the application.

After talking with the experts in the domain we have identified that, to develop and get the supposed power of a micro service application it need to have a well experienced software architects to architect the application.it needs to separate into modules in a well-defined manner because if one micro service do contain irrelevant code modules which needs to be in a separate micro service the decoupling of the application will be compromised and it would result in low performance. But the most dangerous thing is the developers or the architect cannot identity the enlargement of the service after deployment because they need a way to identify the business scenarios and what are the micro services that come together to solve the

problem [8][9]. Unfortunately currently available container orchestration systems suffer from lacking this functionality. So from the proposed system we have planned to eliminate this problem by identifying the business scenarios using machine learning modules and show it to the administrators or the architects of the application where they can decide whether to keep the service as it is or do a refactoring and separate the logic into two or more new micro services.

There are many facts that needs to be taken into consideration when there is a need to the scaling of a micro service application like CPU load, memory utilization, storage utilization, network utilization. These are the absolute measurements which are taken from the server itself but the absolute measurements can give wrong intentions. For examples

The component load can go up when we are deploying a new micro service and the memory utilization can go high because of a memory leak or a bad development. But the existing systems like **kubernetes** is taking only the CPU load of the application to scale the pod as they have named the micro service like that. This is again a place where we go beyond the existing implementations. We will consider about required absolute measurements and we will use measurements from the web gateway using an analytics server and validate the absolute measurements where the system will calculate a weight based measurement using the validated absolute measurements and the relative measurements which will be most appropriate to be used to take the decisions whether to scale up or scale down the application.

One of the main objective of the micro service architecture is to have high availability by using redundant instances of the same micro service. So for these applications there can be load patterns where for a particular season load will be very high and vice versa. When diving into the available applications we can see that they do the scaling in a static manner where they take a performance metric and do the scaling accordingly. But we have identified that this approach has a drawback. For an example we can take a load pattern where the load is increased at the morning and decreased after noon. We will take this as a regular pattern. How this pattern is handled by the currently available systems is by identifying the increment of the load when it happens and increase the instances to manage it handle it. this would not be a

better solution because the requests that will be used to identify the load increment will not consume the redundant instance power.so we came up with a plan to analyze the load patterns and anticipate for the increments or the decrements of the loads, so that the application can do an anticipation scaling to match the requirement. Which is also not available in the current micro service scaling systems.

In existing systems using load balancing, the execution time for each individual micro services are not considered. As time goes on the execution time might increase this might happen when the entries for the database increases or the number of concurrent users using increases. We will consider all of these factors while considering the scaling of the micro services.

1.4.Research Problem

Because of the operational complexity and the lacking of the required automation tools it has become very difficult to implement a micro service application so the smaller companies which do not possess the required skilled human resources, do not go for micro services even if they understand the importance of the architecture and the benefits.

2. OBJECTIVES

2.1.Main Objectives

The main objectives of this research is to create a system which can

1. Manage micro service based applications by letting the application itself to be automatically scaled based on the usage patterns.
2. Manage the orchestration of the micro services after initial deployment.

The above objectives are can leverage the full advantages of the micro services architecture.

2.2.Specific Objectives

The specific objectives which will be considered are:-

1. Gather data from the web gateway for each individual application. This will be the requests that are being sent on to the application. A model will be using reinforced learning to monitor the requests being received and to predict the load for a particular time frame.
2. Create a profiling system that could monitor the applications that are deployed onto the **Docker** individually.
3. From the profiling system it will be necessary to get the business scenarios that are being executed in a sequential manner.
4. The data from the profiling system will be used to predict the CPU load and other properties of the load to the system. The models will again be using reinforced learning for the models.
5. Based on the data gathered the system will automatically scale the **Docker** containers.
6. A Model will be created from the data that is collected from the business scenario models. This will be used for checking the micro service which will need to be most available when the system is deployed.
7. A base model will need to be built which will use supervised learning from the previous models.
8. Build a UI for monitoring aspects of the applications that are deployed onto the **Docker** such as the usage patterns, CPU load etc. This will also include a platform which would make the deployment of micro services very much efficient and easy to achieve compared to existing system.

3. METHODOLOGY

The main functionality of the system has been divided into three components. The sequential manner in which the system will execute is given below:-

1. Analyze the data gathered from the web gateway to predict the request load for the application.
2. Identification of execution patterns inside the Docker based on the requests received and analyze the actual resource allocation for particular request.
3. Using the data gathered, identification of business scenarios that are there in the Micro Services application and manage the redundancy of containers on the Docker.
4. Virtualize the cloud platform in order to demonstrate the product

For further explanations refer below

We will be using a web API for the gathering data of the requests that is directed towards the applications. The applications will be having a snippet which would allow it to send the details of the request every time a user access it. This will be later monitored on the Data Analysis Server which will be provided by the Web Analytics API and will be accessible for our usage. The data will be later gathered and will be used to train our ML Models for predicting the load which would be present at a particular time frame. The ML Models used for this system will be having reinforced learning implemented as this data will change from time to time for each application and this will need to be monitored in order to increase the accuracy of the system.

The data which is provided from the first module will be used for gathering the parameters required inside the system for creating the 2nd ML model which will be used to predict the load which would be present for the Server. In this Model the load would be the I/O usage, Memory Usage, Network Usage amongst others. This will be gathered by monitoring the data being communicated between each individual micro services. The data which is gathered will be used for mainly predicting these parameters. The micro services will be tagged according to their main purpose for example “Authentication” and so on. These data will be collected

from within the system and will be streamed real time for data analysis.

The data gathered from the above module which will give a sequence of the execution plan that is the micro services that are executed. The final module will be used to predict the Business Scenario's which will be executed. While all of these modules will keep track of the tags being deployed for the application, which enable us to make predictions of the loads for each type of application which is deployed on to the system.

The execution plan of the service will be recorded by making use of the tagging system and this data will be stored onto the system. A model will be created which will keep track of all possible execution plans which might occur for a particular service and this will be tracked in order to find the service which is most used, this analysis will be made use of to ensure that the service's availability is high. The collected data will be presented for each individual application since the execution plan for each services differ based on the application's implementation. This model will make use of reinforcement learning to increase the accuracy.

Finally all of these models will need to merge onto a single module which will be the main model from which the system will take decisions [1]. The micro service instance manager will make use of this data for the managing or scaling of the Docker. The cloud environment will need to be simulated for testing purposes at the final stage of the development, as it will not be possible for us to gain access to existing cloud system on which we could test this on. The possibility of using an existing Cloud platform for testing purposes will also be considered if possible.

The applications will be developed by making use of NodeJS. This Scripting language has been selected as it is the most commonly used language for developing micro service applications and it would be easier for developing middleware for the said micro services. The middleware will be used for identifying the services uniquely for collecting data from each of these services.

The developed system will try to get the most advantages out micro service architecture, but it might not be able to utilize the full advantages of the micro

service architecture, until a certain extent. Since the Docker also has some difficulties when developing the full benefit of micro service architecture.

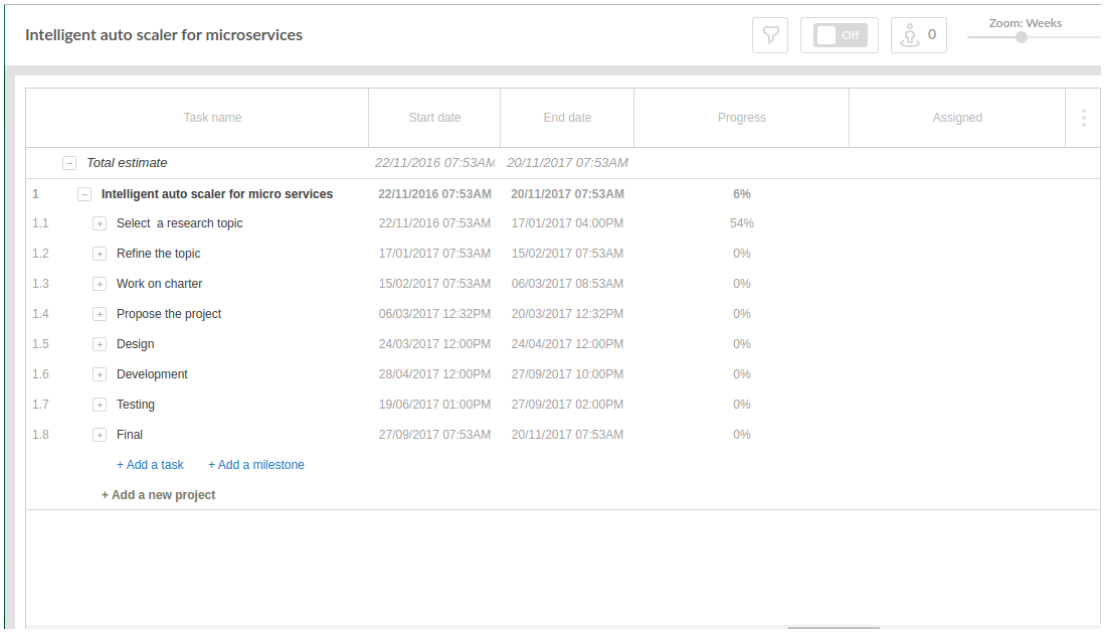

Figure 4: Gantt chart I

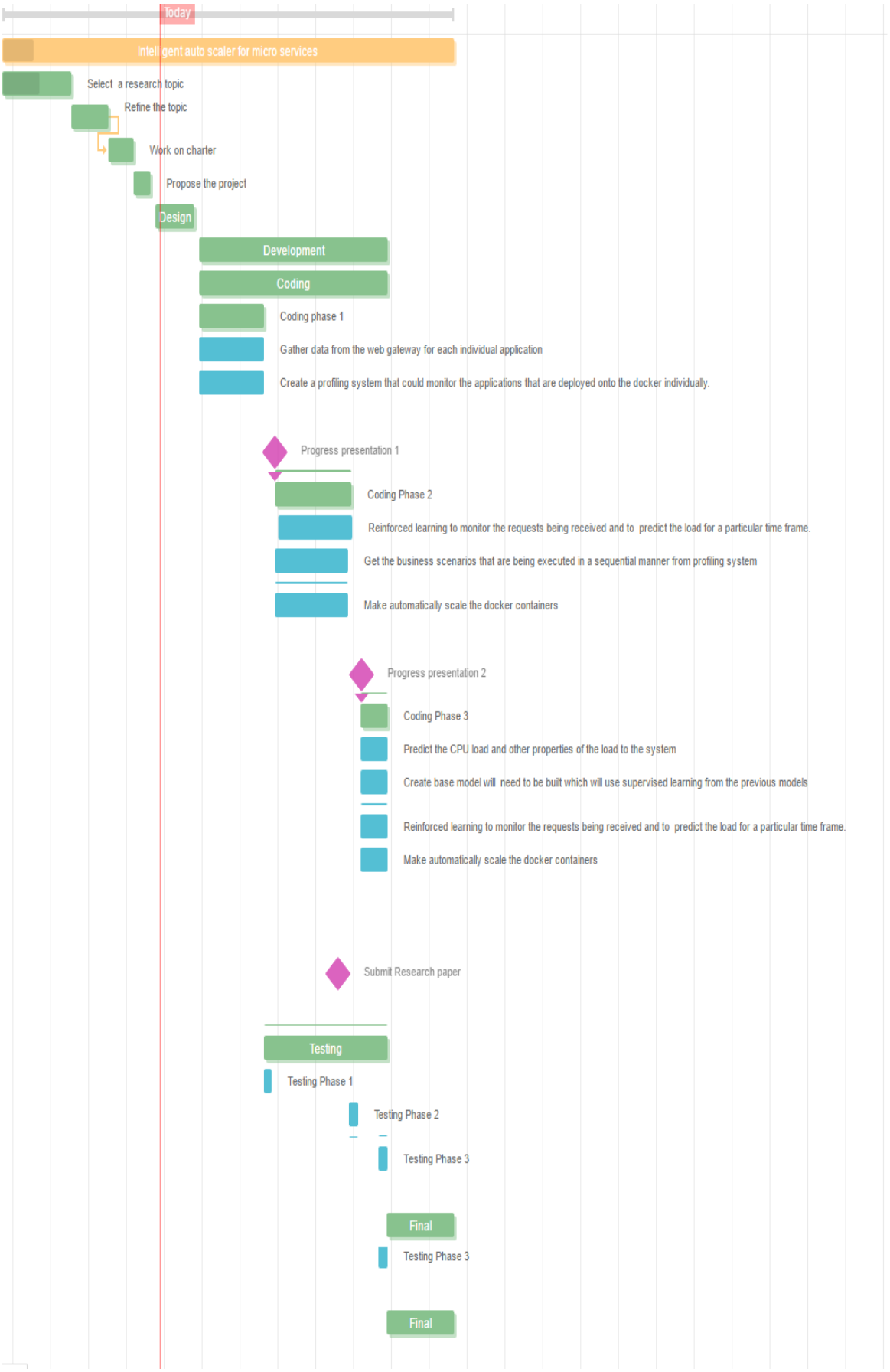

Figure 5: Gantt chart II

4. WORK BREAKDOWN STRUCTURE

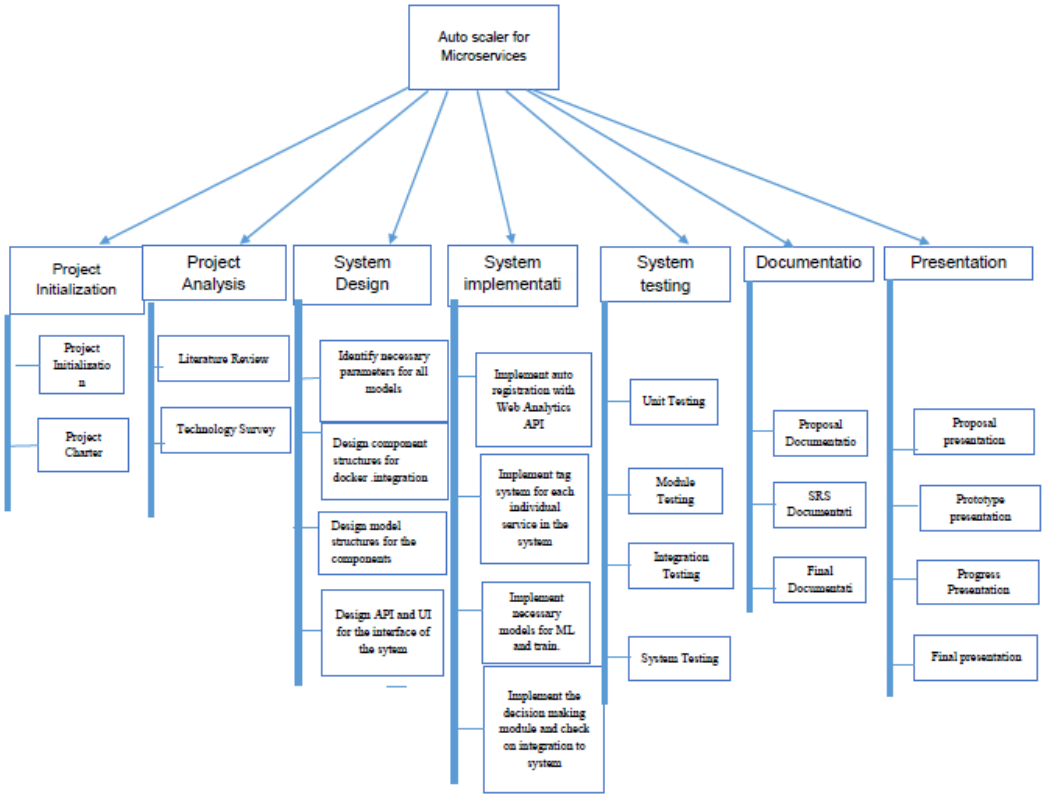

Figure 6: Work breakdown structure

5. DESCRIPTION OF PERSONAL AND FACILITIES

For now we have contacted popular cloud service providers to confirm whether we could deploy our system for testing purposes. At the moment as it seems impossible to do so, we have come to the alternative of using a virtualized environment to simulate a multi clustered cloud architecture for testing out our system.

Table 1: Description of personal and facilities

Member	Component	Task
IT 13148478	Micro service instance managing and decision making module.	● Identify what are the most suitable parameters which should be taken into account to make the required decisions ● Making required decisions which are needed to scale the application using the web API data and individual resource utilization data. ● Managing the micro service instances.
IT 14064432	Web API usage pattern analyzing module	● Register with the web analytics server. ● Get the data from web analytics and stream to the server for analysis. ● Analyze the data gathered from the web API and create a machine learning model to predict the future load
IT 14056826	Absolute resource utilization analyzing and micro service tagging module	● Monitor the business scenario being executed when a micro service is called. ● Identify each micro service and tag them for identification. ● Get the resources which are used when executing the micro service ● Build a model for predicting the resources that are allocated for the micro service.

IT 10230848	Business scenario prediction and analytics visualization module	• Business scenario prediction based on past data • Interface design for the Frontend of the system.

6. BUDGET AND BUDGET JUSTIFICATION

The estimation of the budget required to complete this project is uncertain. If it becomes possible to make use of Amazon Web Services or Microsoft Azure for testing purposes, it would be necessary to purchase an IAAS for testing out our system, otherwise it would not be necessary. If not then it would be necessary to get a separate server for testing out system by virtualizing an environment. Therefore as of now the full expense of this project is unknown. But if an instance were to be purchased

Table 2: Budget and budget justification

Provider	Price
Amazon Web Services	Pricing for Amazon EMR and Amazon EC2 (On-Demand) **Amazon EC2 Price** — **Amazon E** **General Purpose - Current Generation** c1.medium — $0.130 per Hour — $0.030 per c1.xlarge — $0.520 per Hour — $0.120 per cc2.8xlarge — $2.000 per Hour — $0.270 per AWS | Amazon EMR | Pricing https://aws.amazon.com/emr/pricing/
Microsoft Azure	https://azure.microsoft.com/en-us/pricing/calculator/

The instance will be purchased and our own custom docker system will need to be deployed. This will also integrate the individual components built for this project, which is the auto scaling system. The test data will be collected and can be viewed through a REST API which will be developed as part of the project.

7. ABSTRACT PRESENTATION

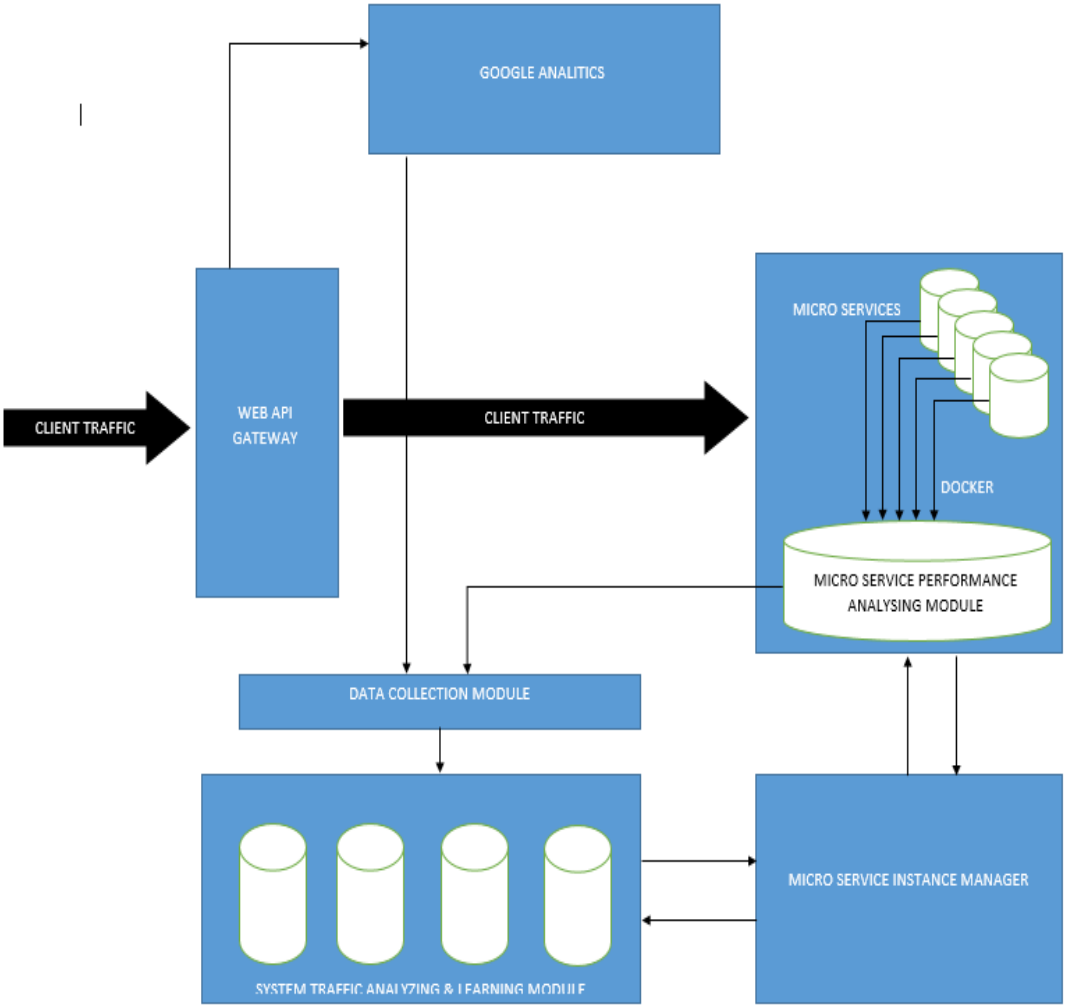

Figure 7: Abstract presentation

8. REFERENCE LIST

[1] Zhen Xiao, Weijia Song and Qi Chen, "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment" IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 6, June 2013.

[2] 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: 27- Mar- 2017].

[3] 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.

[4] How to Think About AWS & Scalability," *A Comprehensive Guide to Building a Scalable Web App on Amazon Web Services - Part 1*. [Online]. Available: https://www.airpair.com/aws/posts/building-a-scalable-web-app-on-amazon-web-services-p1#2-how-to-think-about-aws-scalability. [Accessed: 27-Mar-2017].

[5] 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.

[6] DAAS, A., GUPTHA, I. AND MOTIVALA, A. SWIM: scalable weakly-consistent infection-style process group membership protocol - IEEE Xplore Document In-text: (Daas, Guptha and Motivala) Your Bibliography: Daas, Abinandhan, Indrnail Guptha, and Ashish Motivala. "SWIM: Scalable Weakly-Consistent Infection-Style Process Group Membership Protocol - IEEE Xplore Document". *Ieeexplore.ieee.org*. N.p., 2017. Web. 27 Mar. 2017.

[7] FOWLER, M. martinfowler.com In-text: [5] Your Bibliography: [5]M. Fowler, "martinfowler.com", *Martinfowler.com*, 2017. [Online]. Available: https://martinfowler.com. [Accessed: 27- Mar- 2017].

[8] Hemant S. Mahalle, Parag R. Kaveri and Vinay Chavan (2013, Jan.). Load Balancing On Cloud Data Centres. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3(issue 1), pp. 1-4.

[9] 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.

[10] C.E.Perkins and E. Royer. Ad hoc on-demand distance vector routing. In Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, 1999.

[11] 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.

[12] JARMILLO, D., NGUYEN, D. V. AND SMART, R. Leveraging microservices architecture by using Docker technology - IEEE Xplore Document In-text: [4] Your Bibliography: [4]D. Jarmillo, D. Nguyen and R. Smart, "Leveraging microservices architecture by using Docker technology - IEEE Xplore Document", *Ieeexplore.ieee.org*, 2017. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7506647. [Accessed: 27- Mar- 2017].

[13] 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- Mar- 2017].

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