AUTONOMIC APPLICATION TOPOLOGY REDISTRIBUTION

ERIC RWEMIGABO

Autonomic Computing

MSc Computing Science (Software Engineering and Distributed Systems)
Computer Science
Faculty of Science and Engineering
University of Groningen

[August 29, 2018 at 18:07 – classicthesis version 1.0]

Eric Rwemigabo: *Autonomic application topology redistribution,* Autonomic Computing, MSc Computing Science (Software Engineering and Distributed Systems)

SUPERVISORS:
Vasilios Andrikopoulos
Mircea Lungu

LOCATION:
Netherlands

ABSTRACT

After years of advancement in Cloud Computing, including a significant increase in the number of Cloud service providers within a short period of time, application developers and enterprises have been left with a wide range of choice to fill their need for cloud services. Therefore, with this wide range of choices, one may find themselves making a choice of using a service that is cheaper on paper but ends up costing them more in the long run.
In this project, a system is designed and implemented to cover one of the major concerns to application owners, which is the utilisation of the resources one is paying for. Resource utilisation can be one of the biggest costs to the owner of an application given that it can cost them one of two ways; by the loss of users if the application crashes due to lack of enough resources. And secondly, if the user has to over pay for resources that aren't being used by the application. The system developed uses the MAPE-K automation strategy proposed by IBM. It monitors the user's application and then analyses the application's usage statistics through the provisioning API and thereafter predicts what it's usage is going to be in the next time window. From that prediction, it make an adaptation plan if necessary by selecting a more suitable topology to handle the predicted load and finally redeploys the application with the more adequate Topology. The final system's functionalities are first tested on, a simple voting application and then evaluated using a larger web shop application. Both of these applications use the micro services architecture and the results of the testing and evaluation are to be presented.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

After years of advancement in Cloud Computing, including a significant increase in the number of Cloud service providers within a short period of time, application developers and enterprises have been left with a wide range of choice to fill their need for cloud services. Therefore, with this wide range of choices, one may find themselves making a choice of using a service that is cheaper on paper but ends up costing them more in the long run.

Optimisation refers to the minimisation of allocated resources conditional to keeping the quality of a service at an acceptable level [28]. If the use of the resources one is paying cheaply for isn't optimised, it will end up costing them in the long run through various ways like affecting the performance of their services, over provisioning certain services to mention but a few.

1.1 PROBLEM STATEMENT

One argument for the use of Cloud Computing(CC) from an enterprise perspective is it makes it easier for enterprises to scale their services, which are increasingly reliant on accurate information according to client demand [6]. Given this aspect, we can then go further and conclude that the optimal scaling of their services would be important to them because they would want to get the best utilisation of these services in order to get the most out of their money especially for a start up with a limited budget that would go with cloud computing as a cheaper and more flexible option. In order to achieve the optimal utilisation of resources provided by CC services, the enterprise has to have a way to monitor and analyse how their services use the allocated resources and then make changes if necessary.

A number of projects have been taken on to work on the optimisation of systems with different architectural structures and using different approaches to implement their automation systems. However, many of these systems tackle one or two areas which the system I have developed for this project covers. In this project, I design and develop a component-based architecture, which uses MAPE-K loops introduced and explained in a following chapter to perform the optimisation of the cloud based application with a micro service style architecture that it is deployed on. This strategy ensures the coverage of the missing areas like the implementation of a plan and Knowledge base component that can make use of other data otherwise unavail-

able or unused by the automation system and hence try and make better decisions in the automation of the application being monitored.

1.2 PROJECT DETAILS

As seen in the paper [3], a CBA Lifecycle is proposed, which can be viewed as a set of MAPE-K loops [20] shifting between the defined architectural models (alpha-topologies). These shifts are caused by controllers that provide coordination across the different stages of the lifecycle and in order to validate this, an IDE in the manner discussed by the MODAClouds approach [5] is required therefore for field study validation of the lifecycle through collaboration with the industry.

The system I have developed is meant to realise the for the above mentioned proposal through the implementation of the back end functionalities by the use of the aforementioned MAPE-K loops. MAPE-K stands for Monitor, Analyse, Plan and Execute by using Knowledge about the system's configuration and/ or including other information like the historical data. For the implementation of my system, I develop each of these as separate components, which interact with oneanother in order to complete this loop. This system's loop is run on top of a cloud container application and uses the available APIs from the container provider to monitor and record statistics of each of the containers in the particular micro service. These statistics are the start point at which the cloud application can be monitored and then optimised by switching between the different topologies provided by the owner of the application (System user). A switch occurs if the application topology doesn't meet the service level objectives specified by the owner of the application and it is either under using or over using the resources provided to it hence costing the application owner either financially or in terms of loss of users due to poor response times.

1.3 DOCUMENT ANALYSIS.

In the chapters that follow; Chapter 2 starts off with a literature review, which covers some of the fundamental concepts that make up the project hence providing some insight into Autonomic Computing. I then go ahead and provide the details of the system including the full system's requirements and additionally looking at the requirements of the individual components of the system in detail.

In Chapter 3, I provide the important design documentation of the system. I start by presenting some of the design patterns used for the system to achieve communication between the various components and other useful aspects that help the system achieve its complete

functionality. Additionally, the full system's architecture is also provided hence concluding the chapter.

Chapter 4, provides the some of the other details of the system by looking at the Technologies that help the system be able to accomplish the different tasks involved in the MAPE-K process and additionally provides the details of each of the MAPE-K components and finally how they interact with each by providing the full system's Architecture I then close off the chapter by describing some of the important algorithms used in the project.

Chapter 5 presents the Evaluation of the system this will introduce the applications that were chosen for testing my system, and then go further to include the different battery of tests run in order to confirm the functionality of the components of the system. The results of these tests will be presented and then data presented about the complete system's functionality.

Finally, in Chapter 6, I present the final review of the complete system by providing details of the various functionalities that I was able to implement and additionally, each of the unimplemented functionalities. This chapter is then closed off with a proposal of some of the future work that can be done in terms of both the extension of this system and in the field of Autonomous Application Topology Redistribution.

2 BACKGROUND AND RELATED WORK

In this chapter, a review of the literature relevant to the development process of this application and other relevant terms to help with further understanding of this project are to be presented. Further more, we take a look at the MODA Clouds project, which is another could application automation system, which uses the MAPE strategy and a few other systems developed that implement and test some of the individual components developed in this system.

2.1 BACKGROUND

This section covers some background knowledge into the work done in the research areas related to this work, including terms constantly used throughout this report, which help to drive this project.

2.1.1 *Topologies*

The concept an application topology, which based on the article The optimal distribution of applications in the cloud [4] in summary is a labelled graph with a set of nodes, edges, labels and, source and target functions. It introduces and explains in full detail the concept an application's topology. A Topology can be referred to as a μ-Topology, which is split into α-Topologies, and γ-Topologies, concepts which are also explained in the article mentioned above. The article also introduces the concept of viable topologies. The focus of this section will be on making clear what the α-Topologies and the Viable topologies are as these are the concepts most relevant to the system's development. Therefore, these are discussed further below including the relation to the project.

2.1.1.1 *α-Topologies*

As can be seen from an example in the article [4], a type graph for a viable application topology can be referred to as a μ-Topology and therefore, the α-Topology is the application specific subgraph of the μ-Topology, which refers to the general application architectural setup. The relation and importance of the above to the target functionality of the system being developed, which is to automatically switch between the available viable topologies provided by the user, is that it provided knowledge on what the system should be able to expect as input in terms of the topologies that will be used to determine the

best option for the optimisation of the application the system is run on.

2.1.1.2 *Viable Topologies*

Knowing about the α-Topologies, it becomes clear what the term viable topologies refers to in the context of this project, are the different suitable topology options that will be available to the automation system being developed in order for it to be able to select the best topology option, which follows the set Service Level agreements and therefore is the best option for the particular application usage scenario. The concept of viable topologies is important for the development process of this automation project because it is a requirement for the system that the application it will be run on should have a number of viable topologies defined by the system architect, which will be the topologies the system switches between to optimise the resource usage of the application. The creation of the viable topologies for the distribution of an application is out of the scope of this project however, there are a number of works available to the developers, which can help with this process. [7] for example provides a solution to help developers ensure portability of their applications and [16] presents a solution, which enables the automatic derivation of provisioning plans from the needs of the user among other papers in the field.

2.1.2 *Auto-scaling (Autonomous Computing)*

Different researchers have delved into a number of projects using different strategies to try and solve a variety of problems in the field of automation. These different projects range from: *The monitoring of different metrics of the system*, For example whether to monitor the lower level metrics like the memory, cpu usage or even the network statistics or the higher level metrics like the rate of response to requests of the system being optimised. *To the type of analysis strategy used to determine whether to scale up or down* for example the use of a predictive methods (see [23]), reactive or rule based approaches (see [1]) or hybrid methods using both reactive and predictive approaches. These different projects and automation strategies are discussed in a survey [27], which helped provide an insight into the different options available to help with the completion of the different components of the system. The strategies used in the project will be looked into in a later chapter.

2.1.3 *Control Loops (MAPE-K)*

Throughout the various papers discussed in the survey [27] in the previous chapter, it is noticeable that the MAPE control loop strategy

is a commonly used automation strategy, which involves the use four main procedures that make up this strategy, which are Monitor, Analyse, Plan and Execute. The MAPE automation strategy [20] used for this project is complimented with a Knowledge base, which all the MAPE components interact with in order for the system to make an informed optimisation decision. These MAPE control loops including the knowledge base are the core driver of the system being developed as they are what makes up the complete functionality of the system, which are further looked at in a following chapter including the implementation process of the autonomous system.

2.1.4 *Microservices Architecture*

Microservices as discussed by Martin Fowler [24] describes an architecture style of building systems into a suite of smaller services each running on their own. These may be written in different programming languages and use different data storages. The microservice architecture is the architecture style focus for this project in terms of applications that the developed autonomous system will be able to perform its optimisation services on. The isolated services in this architecture style make it possible for the developed system to be able to individually run its loops on each service and therefore in the end perform the full assessment of the whole system, therefore performing its optimisation more efficiently and it's because of this characteristic that the Microservice style architecture was selected for this project. Additionally, the microservice style architecture is also well supported by most of the containerisation engines. This is an advantage in the case of this project since the popularity of containers among cloud developers recently has increased and therefore I was able to easily find a number of test applications that have the micro service architecture style and were deployed in containers especially the one selected for this project (Docker), which is introduced and shown later in a following chapter.

2.1.5 *Fuzzy logic*

Fuzzy logic is a concept that has been used a lot to help machines loosely translate human terms like high and low into concrete values that they can use to assess a certain situation. The paper [32] points out that the fuzzy logic concept stems from the Computing with Words methodology, which all started from [31]. Fuzzy logic helps in simplifying the decision process of complex systems and therefore, it presented as a compelling option for the implementation of the analysis phase of the MAPE loop, where it would help decide whether the application being managed by the autonomous system to be developed required a switch in topology or not. The

decision to use fuzzy logic was further enforced by [15], where they used fuzzy logic to help optimise the scaling mechanism of a cloud service. Finally, the availability with the help of a standard for fuzzy control programming in part 7 of the IEC 6113 published by International Electrotechnical Commission, I was able to learn the basics of the language and use it with jFuzzyLogic [8], which is introduced in Chapter 4.

2.2 RELATED WORK

As mentioned previously, a number of projects have been undertaken in order to tackle various problems in the field of self adaptive systems. A lot of the projects in the field of automation have more specifically covered particular domains similar to robotics and smart house systems. Even though these also help provide some understanding because of the work done there, my focus is in the cloud computing domain where there are a number of techniques that have been employed to perform the task of automation among the various projects that have been undertaken. Many of these works look to tackle, solve or answer particular questions in the cloud computing field. These different solutions for the particular problems proved to be an important resource because by looking through and combining these works, I was able to tailor a solution for the various components in the MAPE-K loop.

Some projects looked into the different ways to most effectively estimate the required resources to be made available. [1, 2, 9] all implement Rule-based approaches, which is an approach of resource estimation where; if, and else rules are used in order to trigger a particular resource provisioning function. Additionally, work done in works like [14, 17, 30] provide predictive solutions to the resource estimation problem done in the analysis component of the system to be developed. They performs the predictions by monitoring and using either lower or higher level metrics of the managed application of which the lower metrics would represent the cpu, network memory and so on whereas the higher level metrics would represent a metric like the response time of the application [27]. This range of solutions, which includes others helped with the decision to use a predictive (*regression*) solution for the analysis component of the system with a combination of a fuzzy logic solution introduced in the previous section and used by [15].

One of the major problems associated with the auto scaling of an application is oscillation, where the auto scaler in this case would switch to a new viable topology and within a short time switch that topology again[27]. The solution of the use of dynamic parameters as seen for example in [21] helped provide inspiration to use a solution for the planner where the switch of a topology is only authorised

when the time after the last switch is double the time it takes for the change (redeployment) to be executed and hence mitigating the oscillation problem. Finally, the work done in [4] provides insight into the selection of a topology among the available alternative topologies provided and with this information among others, the implementation planning component was made possible.

3 SPECIFICATION AND DESIGN

In this chapter, I present the requirements specification and design of the project in two sections. In the first section, some of the most important functional requirements of the full system are presented and then, the functional requirements of the individual components of the system are presented. After this, some of the use cases of the system as a whole are presented in the following section and including some design patterns after which the logical view of the system is presented in two images, which depict the interactions of the components and the functions performed by these components.

3.1 REQUIREMENTS SPECIFICATION

Some of the requirements presented in [18] provided a template upon which to start writing my system's main functional requirements, as they cover the functional and non-functional aspects to enable the dynamic (re-)distribution of applications in the cloud, which is the aim of my system. However, during the development of the system, a few other functional requirements were realised and added to the list for both the system and its individual components.

3.1.1 *System's Functional requirements*

FR1 The System should be able to connect to or interact with a containerisation engine.

FR2 The System should have access to the containerisation engine's API.

FR3 The System should sort the services of the application being manage into a list.

FR4 The System should have access to the alternative viable topologies.

FR5 The System should be able to Monitor data from the managed Application.

FR6 The System should be able to Analyse the data from the managed Application

FR7 The System should be able to make a Plan using the data from the analysis of the managed Application.

FR8 The System should be able to Execute the plan made for the managed Application

FR9 The System should create Monitors for each of the services in the managed Application.

FR10 The System should create Analysers for each of the services of the managed Application.

FR11 The System should have access to the Service Level Agreements or Service Level Objectives set by the user.

3.1.2 *Component Requirements*

3.1.2.1 ***Monitor Component***

FR11 The Monitor component should create sensors for each of the containers of the service it is monitoring.

FR12 The Monitor component should log statistics for each of the containers of the service being monitored.

FR13 The Monitor component should set the sensors to monitor different application metrics.

FR14 The Sensor(s) should check that the metric values recorded are within the SLOs.

FR15 The Monitor component should notify the analysis whenever a new statistic is recorded.

FR16 The Monitor component should notify (differently/ Urgently) the analysis whenever a metric out of scope of the set SLA/ SLO is recorded by the sensor(s).

FR17 The Monitor component should save the monitored statistics to the knowledge base.

FR18 The Monitor component should continuously monitor the service until the point that it is no longer available.

3.1.2.2 ***Analysis Component***

FR21 The Analysis Component should receive notifications from the Monitor component of new statistics.

FR22 The Analysis Component should perform data analysis in time windows defined by the user.

FR23 The Analysis Component should retrieve a batch of statistics in a given time window for analysis from the knowledge base.

FR24 The Analysis Component should perform a set of analysis tactics on the statistics gathered from the knowledge base.

FR25 The Analysis Component should provide a visual representation of the analysed data.

FR26 The Analysis Component should perform a prediction of data points for the next time window.

FR27 The Analysis Component should make an estimation of the resources that need to be added, removed or kept as is for each of the services.

FR28 The Analysis Component should make an aggregation of the results from the various analysers.

FR29 The Analysis Component should create a topology suggestion using the estimated resources based on the topology running and the aggregated results.

FR210 The Analysis Component should notify the plan component of the new topology suggestion.

FR211 The Analysis Component should save the results from the analysis to the knowledge base.

FR212 The Analysis Component should save the topology suggestion to the knowledge base.

3.1.2.3 *Plan Component*

FR31 The Plan Component should be able to receive notifications from the Analysis Component.

FR32 The Plan Component should have access to the list of alternative viable application topologies.

FR33 The Plan Component should have access to the recommendation(Adaptation request) from the Analysis Component.

FR34 The Plan Component should ensure enough time (set by user) has passed since last topology (re-)distribution.

FR35 The Plan Component should retrieve the latest suggested topology by the Analysis component.

FR36 The Plan Component should make a comparison between the suggested topology from the Analysis Component to the alternative viable application topologies.

FR37 The Plan Component should retrieve the alternative viable topology closest in similarity to the suggested topology.

FR38 The Plan Component should have access to a record of the currently running or deployed topology.

FR39 The Plan Component should notify the Execution component when a change is confirmed.

FR310 The Plan Component should store the new topology for the (re-)distribution to the Knowledge Base

3.1.2.4 *Execution Component*

FR41 The Execution component should be able to receive notifications from the Plan component.

FR42 The Execution component should create an Effector, whose job it is to perform the execution actions.

FR43 The Execution component should have access to the alternative topologies available.

FR44 The Execution component should have access to the containerisation API.

FR45 The Execution component should be able to run the commands necessary to make the topology change requested by the plan component.

FR46 The Execution component should confirm when the change has been made successfully

FR47 The Execution component should record or save the time of the latest change of topology.

FR48 The Execution component should record the time it took to make the change and update it in the knowledge base

FR49 The Execution component should be able to reset the MAPE-K loop to start working on the newly redistributed topology.

FR410 The Execution component should update the currently running topology.

FR411 The Execution component should save the new topology change to the knowledge base.

3.1.2.5 *Knowledge Base Component*

FR51 The Knowledge Base Component should provide an access point for the various components to create the necessary data.

FR52 The Knowledge Base Component should provide an access point for the various components to update the necessary data.

FR53 The Knowledge Base Component should provide an access point for the various components to retrieve the necessary data.

FR54 The Knowledge Base Component should provide an access point for the various components to delete data.

FR55 The Knowledge Base Component should contain a record of the alternative viable topologies.

FR56 The Knowledge Base Component should contain a record of the statistics recorded by the monitor component.

FR57 The Knowledge Base Component should contain a record of the data output by the analysis component.

FR58 The Knowledge Base Component should contain a record of the topology picked by the plan component to be deployed.

FR59 The Knowledge Base Component should contain a record of the successful topology changes made by the execution component including the time.

FR510 The Knowledge Base Component should contain a record of the service level objectives set by the system user.

FR511 The Knowledge Base Component should contain a record of the other user preferences like the time windows for the analysis e.t.c.

FR512 The Knowledge Base Component should contain a record of the history of the performance of the various topologies that have been run before.

3.2 DESIGN

In this Section, I presents 2 use cases, which are cases under which the system is expected to behave differently. In the first case, the system records normal workloads under which the managed application should not have any problems dealing with and therefore, there is no need for a change. In the second use case, the system predicts stress on some of the services of the managed application or under use of the resources and therefore requires an appropriate change to be made in order for the application to utilise it's available resources optimally.

3.2.1 *Use Cases*

Use Case 1	**Normal Managed Application load**
Goal:	This use case depicts a situation where the managed application load is predicted to be within the Service Level Objectives defined by the application owner. With this case, the system is expected to keep reporting the normal application usage and not make any changes to the topology
Pre-Condition:	The managed application is running, the system is deployed on top of it and is monitoring the usage statistics of the application
Post-Condition:	The System has run an analysis of the statistics and detected that no changes are required and hence, there are no changes made and the monitoring and analysis process continues.
Primary Actor:	Managed application

Assumptions:

- Alternative Viable topologies are made available by the application owner.
- The Service Level Objectives are defined by the application owner.
- The application services are running normally
- The application traffic is predicted to be on par with the resources made available to the application.
- The User has set other parameters like the preferred time windows for analysis.

Main Success Scenario:

1. The system deploys its MAPE-K control loop on the application services.
2. The system gains access to the containerisation API.
3. The monitor functionality records the statistics received from the containerisation API and reports to the Analysis.
4. The analysis component makes a record of the time of the first notification/ statistic from the monitor component.
5. The analysis component compares the current system time with the recorded time every time it is notified.
6. After the correct (set by the user) time window has passed, the analysis component makes a record of that last timestamp and performs an analysis action on the data of the time window.
7. The analysis component makes a prediction of the expected workload in the next time window and suggests the number of containers required to be added or removed
8. The number to be added or removed is 0 and therefore no further actions are required.
9. The analysis component continues to compare the current time to the last recorded timestamp for the next analysis window

---

Extensions (Temporary Spike):

3a The Monitor functionality reports high metrics and shortens the time window to the analysis.

6a Analysis is performed on the shorter time window and predicts that increased use was a temporary spike so, no additional resources should be provisioned.

Table 3.2: Use Case

Use Case 2	**High/ Low Managed Application load**
Goal:	This use case depicts a situation where the managed application load is predicted to be outside the Service Level Objectives defined by the application owner. With this case, the system is expected to make a change in the topology being used by the managed application and therefore get the application back within the Service Level Objectives set by the user.
Pre-Condition:	The managed application is running, the system is deployed on top of it and is monitoring the usage statistics of the application.
Post-Condition:	The System has run an analysis of the statistics and predicted values outside the SLOs and therefore a change in topology is performed and a new topology is being used by the managed application.
Primary Actor:	Managed application

Assumptions:

- Alternative Viable topologies are made available by the application owner.
- The Service Level Objectives are defined by the application owner.
- The application services are running normally
- The application traffic is predicted to be outside the set SLOs within the next time window.
- The User has set other parameters like the preferred time windows for analysis.

Main Success Scenario:

1. The system deploys its MAPE-K control loop on the application services.
2. The system gains access to the containerisation API.
3. The monitor functionality records the statistics received from the containerisation API and reports to the Analysis.
4. The analysis component makes a record of the time of the first notification/ statistic from the monitor component.
5. The analysis component compares the current system time with the recorded time every time it is notified.
6. After the correct (set by the user) time window has passed, the analysis component makes a record of that last timestamp and performs an analysis action on the data of the time window.
7. The analysis component makes a prediction of the expected workload in the next time window and suggests the number of containers required to be added or removed
8. The number to be added or removed is greater or less than 0 and therefore a change in topology is required.
9. The Analysis component makes an addition and/ or subtraction to the resources available in the current topology therefore coming up with a recommendation of a topology like structure for the required resources
10. The recommendation is saved and the Plan component is notified.
11. The Plan component checks the time of the last Topology change.
12. If enough time has passed since the last Topology change, the Plan component accesses the alternative topologies in the knowledge base and selects the one closest to the suggested topology
13. The Plan component notifies the execution component of the required change and saves the suggestion.
14. The Execution component retrieves the selected topology and runs the required commands to redeploy the application in the new topology.
15. The Execution component updates the time to redeploy the application.
16. The execution component updates the last successful redeployment time and resets the system to run on the new topology.
17. The loop restarts

Extensions (Reactive):

3a The Monitor functionality reports high metrics and shortens the time window to the analysis.

6a Analysis is performed on the shorter more urgent time window.

7a A prediction is made and the prediction is outside the SLO.

10a Plan is notified with an urgency/ priority recommendation.

11a No time check is performed.

Extensions (Oscillation Mitigation):

11a Not enough time passed triggers wait action (Depending on how close to breaking point system is predicted to be).

Table 3.4: Use Case

3.3 SYSTEM ARCHITECTURE

3.3.1 *Activity Diagram*

In the diagram below, the activities required to perform the functionalities in the particular component instances are presented. The monitor and analysis components have multiple instances and are managed by the monitor manager or the Analyser manager and therefore these managers ensure the aggregation of data for the entire application topology being managed. The details of the manager and individual components shall be discussed in the following chapter. At the start of the system, each of these components are created by their managers and their activities after that are as follows.

3.3.1.1 *Monitor*

A monitor component is created for each container in the particular service being monitored. This component starts off by creating sensors for each of the metrics it is meant to monitor for that container and then the monitoring task begins. Every 2 seconds, the Sensors send their registered metric to the monitor component and these are metrics are collected to form a statistic. A sensor can also report a metric which is outside the SLOs and this will cause the component to create a new more urgent statistic. After the statistic is created it is stored to the Knowledge base and the Analyser is notified.

3.3.1.2 *Analyse*

An analyser is also created for each of the containers of a service therefore, the analysers and monitor components have a one to one relationship. When an analyser is created, it waits for it's first notification from the sensor it has a relation with and when it receives it, the analyser takes note of the time in the first statistic the sensor saved. After this point, the time window is checked every time a notification is received and when enough time has passed, the components retrieves the statistics for that window, sets the last timestamp as the latest time and analyses the data. On the other hand, an analysis is also performed on double time windows to get a more clear analysis with more data. After the analysis is done, the manager aggregates the data and notifies the plan if necessary of a necessary change.

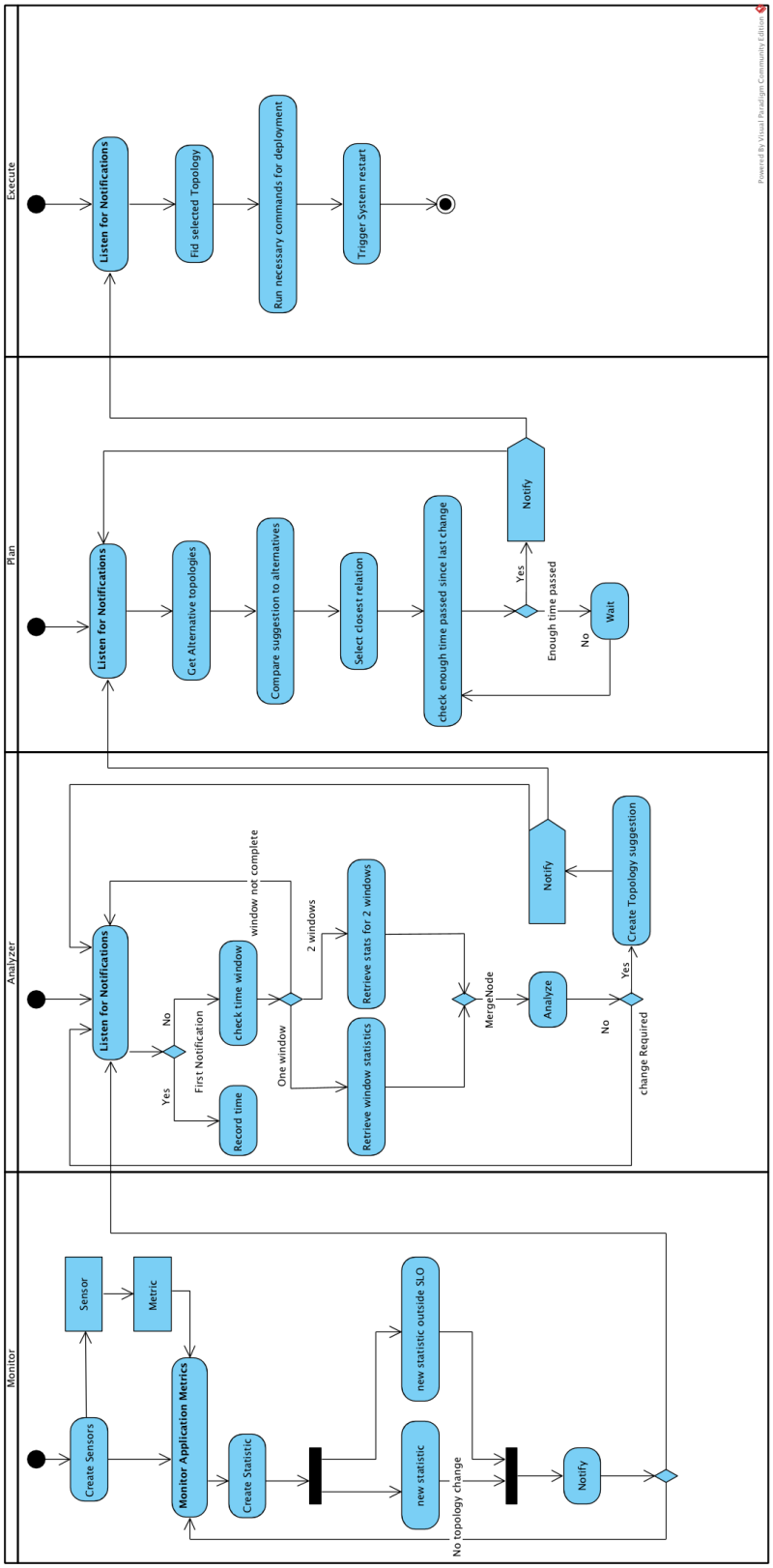


Figure 3.1: Activity Diagram

4 IMPLEMENTATION

In this chapter, the details of the implementation of the system are presented including a description of the individual components of the system and how they interact, create and share data, some of the technologies that were helpful and necessary to the implementation of the system's components including their use to the system.

4.1 TECHNOLOGIES AND THEIR UTILISATION IN THE SYSTEM

4.1.1 *Containerisation Technology (Docker)*

Containerisation[26] is a lightweight virtualisation technique, and virtualisation is necessary for the deployment of applications on the cloud.Therefore, when starting the development of this system, I had to select a containerisation engine where I could access the necessary metrics and upon which to deploy the test cloud applications given that the aim of the autonomous system would be to manage the resources of an application on the cloud so, I had to simulate this environment. There are a few containerisation technologies like rkt[29] developed by CoreOS, Solaris Containers[25] developed by Oracle and so on. However, given that I have some past experience with the Docker Engine, and additionally, the significant number of applications developed for and deployed with the docker engine, the choice was quite clear on what containerisation technology I should use for my project.

4.1.1.1 *Docker API*

While I had used the Docker engine before for the deployment of an application, I had not used it for the development of a system and therefore, I had to looked into its Application Programming Interface (API). From the page [10], I found that it has a Software Development Kit(SDK) for Python and Go, and number of unofficial libraries for a number of programming languages, which opened up my options on the programming language I could use and enforced my confidence in the selection that had been made.

4.1.1.2 *Docker Compose*

One additional advantage of the docker engine is its easy compatibility with the Micro service architecture[24]. Docker Compose[12], a tool for running applications with multiple containers provides

this feature when defining the different components in the docker-compose.yml file during the set up of the application. Given that, a number of micro service applications have been developed and deployed on docker, which provided me with a number of options both simple and complex for the testing phase of my system's components.

4.1.2 *Programming Language (Java)*

After deciding on the containerisation technology to use, it was time to move on to the programming language and my extensive knowledge in the Java as compared to other languages was a starting point. Next on the checklist was the compatibility with Docker and as seen from [10], there is an official library for Java, which added to my confidence in the choice. Finally, the extensive number of options available to me in terms of what database to use also pushed my decision towards the java programming language.

4.1.3 *Database (Relational)*

The option for a database was between either a Relational database or a NoSQL database and I was drawn towards the use of a relational database because of more previous experience with relational databases and they would properly accommodate the data that would be passed through the system.

4.1.4 *Fuzzy Logic (jFuzzyLogic)*

jFuzzyLogic is an opera source Java Library for Fuzzy Logic, which was introduced in Chapter 2. The Library was created to aid in the programming of Fuzzy Logic control systems using the standard Fuzzy Control language defined in the IEC 61131. Using the paper [8], I was able to get some understanding of the library and some useful knowledge on how to use it in my project's development and specifically the analysis component.

4.1.5 *Locust IO*

[22]

4.2 APPLICATION COMPONENTS

In this section, I discuss the individual system components of the system. Using [19] as a resource during the planning development phase of this system, I was able to implement the components even if some

of the behaviour templates were not applicable with this system's domain.

4.2.1 *Sensor*

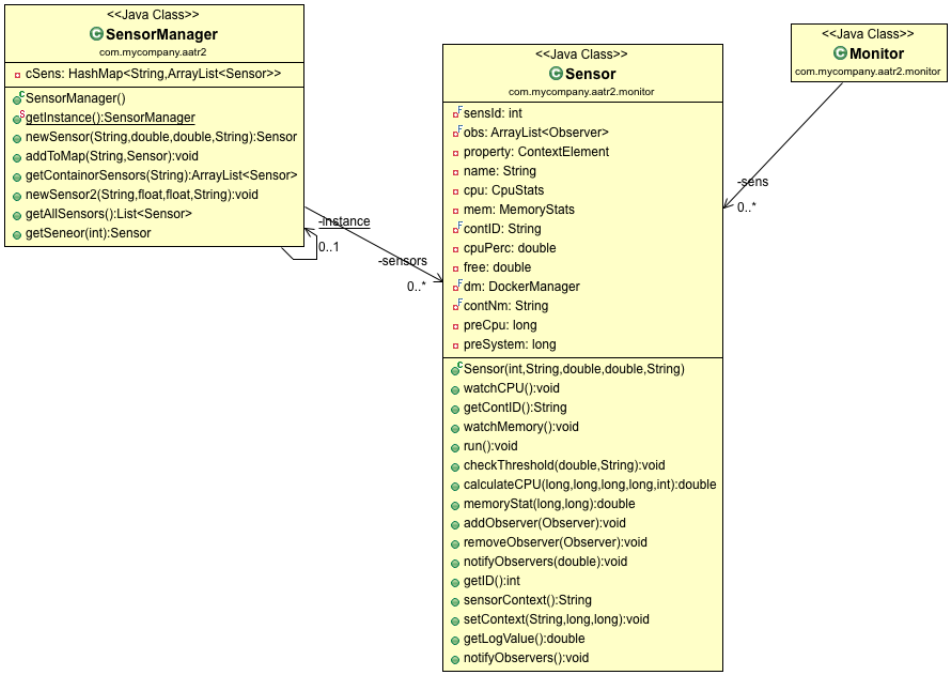


Figure 4.1: Sensor Component Class Diagram

4.2.2 *Monitor*

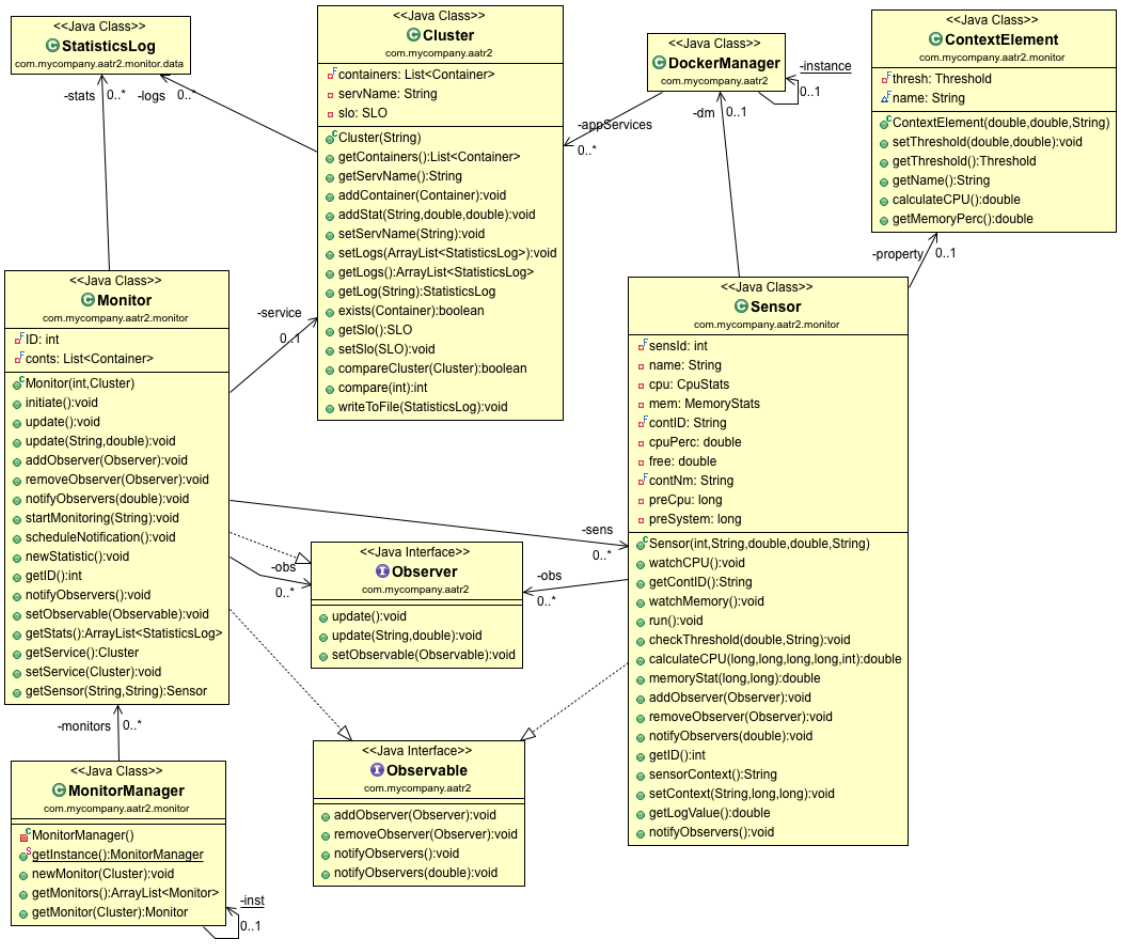


Figure 4.2: Monitor Component Class Diagram

4.2.3 *Analyse*

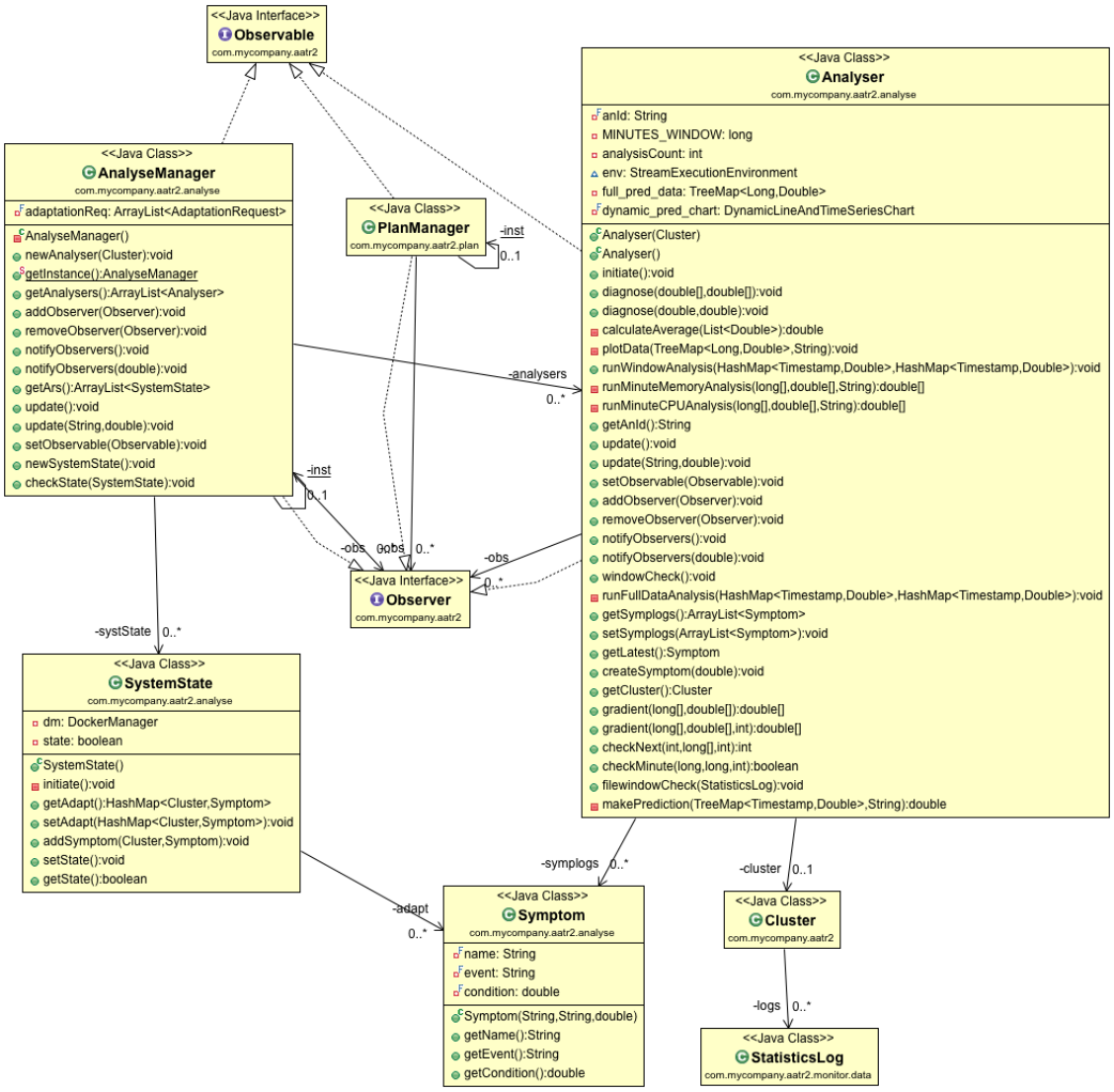


Figure 4.3: Analyse Component Class Diagram

4.2.4 *Plan*

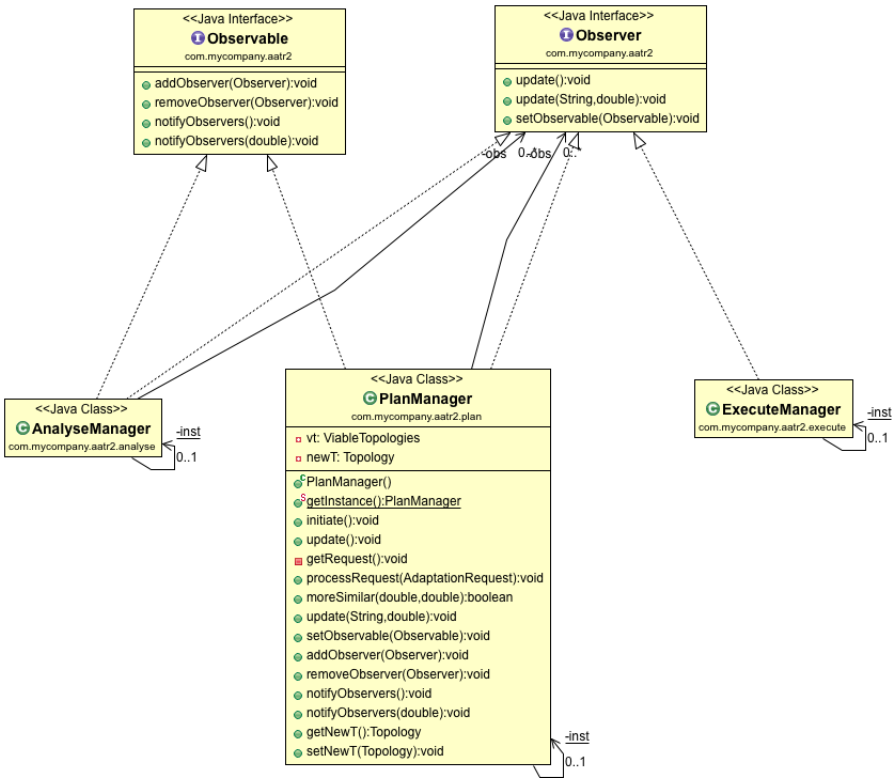


Figure 4.4: Plan Component Class Diagram

4.2.5 *Execute*

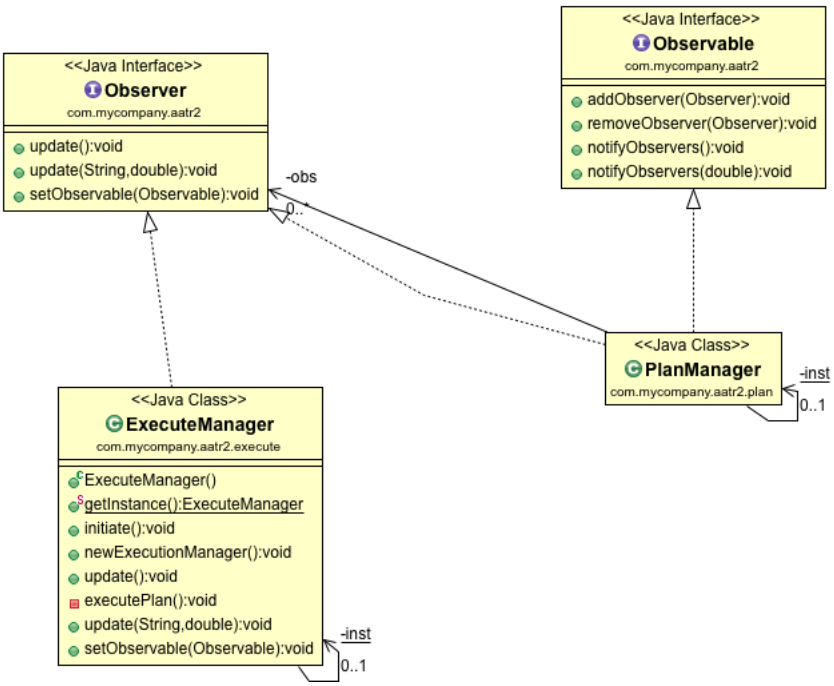


Figure 4.5: Execute Component Class Diagram

5 TESTING

During the development of the system, there were a battery of tests run on each of the individual components to ensure their functionality and furthermore their interaction with each other when integrated in order to ensure that they work together. However, this chapter presents the more important tests run on the more complete versions of the system. After implementing three of the major components of the system, that is the Sensor, Monitor and the Analysis components, the more important phase of testing begun and in the following sections, I present the results of most of these tests and finally, an evaluation of the results of the System is done in the following chapter.

5.1 TEST SUITE

5.1.1 *Test Application*

The testing of the system was done using two different applications assembled from Github. These applications include:

example-voting-app:
After the initial testing of my system's Monitor and and Analysis components and confirming that they performed the basic functionality, I needed to test the more complex features of the Analysis component, which were the prediction and recommendation functionalities. To test these, I needed a simpler more basic application where I would be able to quickly write a testing script for the load simulator application I was going to use so, I decided to use [13] Example Voting App. It is a simple voting application where a user either vote cat or dog. I also decided to use this application because I was having problems with the login feature that was on the application introduced in the section above so, to save time, I changed applications and found this second micro service docker application among the other available options on Github.
The services in this Application include the *voting-app* service, where the users case their votes, *result-app* service, which is where the user views the vote tally percentage, which is retrieved from the database, *redis* service, a queue to handle the votes coming it, *worker* service to process the voted and send them to the final database service *db* service.

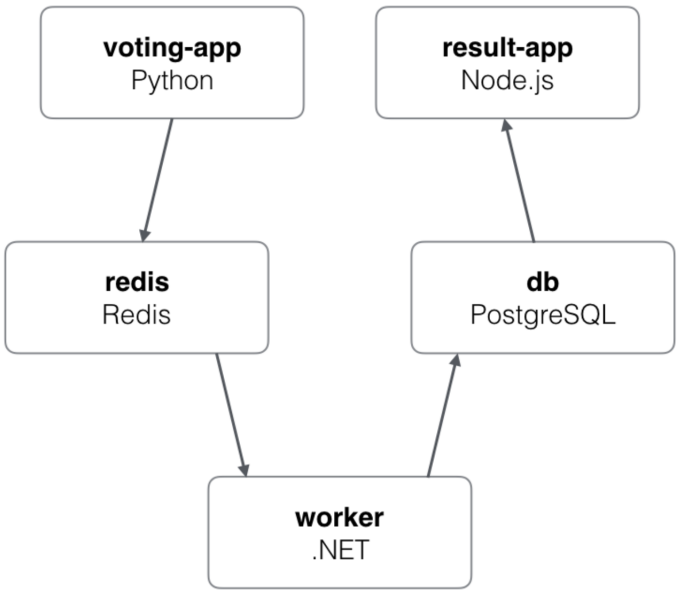


Figure 5.1: Architecture of the test application

5.1.2 *Load simulator*

Locust IO[22], the Load testing application introduced in Chapter 4 was used for to load test the test applications. I wrote the test scripts for the test applications for example the short test script from the load testing done on the Example Voting App is presented in Listing 5.1. By running this script, the simulated user simply initially randomly performs a vote and then randomly changes the vote every time locust sends a request by the simulated user. Listing 5.2 provides the ports where the simulated users can connect.

```
from locust import TaskSet, task
import random
class MyTasks(TaskSet):
    # vote = null
    # vote function
    def vote(self, vt):
        self.client.post("/", {'vote': vt})

    # initial random vote between cats or dogs
    @task(2)
    def votecat(self):
        self.vote("a")

    # change vote task
    @task(3)
    def votedg(self):
        self.vote("b")
```

Listing 5.1: Example Voting App testing Script

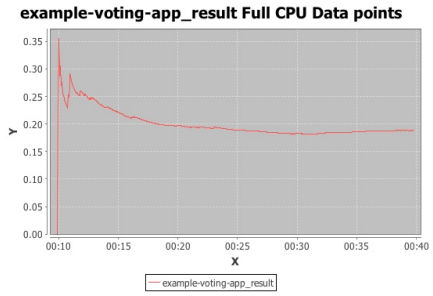
```
from locust import HttpLocust
from MyTaskSet import MyTasks

class MyLocust(HttpLocust):
    task_set = MyTasks
    min_wait = 5000
    max_wait = 15000
    host = 'http://localhost:5000'
```

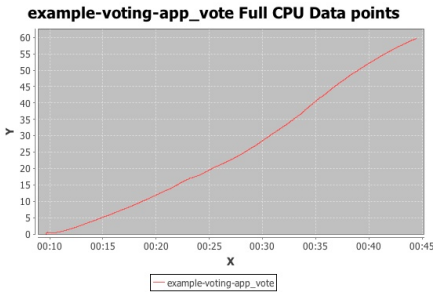
Listing 5.2: Locust open ports Script

5.2 SENSOR, MONITOR AND ANALYSIS COMPONENT TESTING.

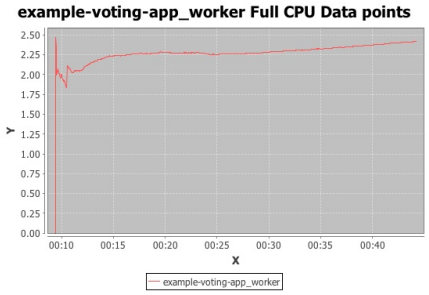
As seen in the Component diagrams in Chapter 4, the These components work closely together and in order to fully test one of them, I had to have all of them working. For the testing phase of these components, I started out by testing the Sensor connection to the Docker API to see whether the statistics metrics were being accurately monitored by the sensor and to confirm, I compared them to the container statistics produced when running the command *docker stats* in the terminal. After confirmation of the accuracy of the sensor statistics, I moved on to the monitoring component. Since there is a monitor instance for each of the services being monitored, I decided to plot graphs for each of the statistics for each of the metrics. Therefore creating 2 plots per container per service. Figure 5.2 shows a sample of the statistics recorded from testing the Example voting application.



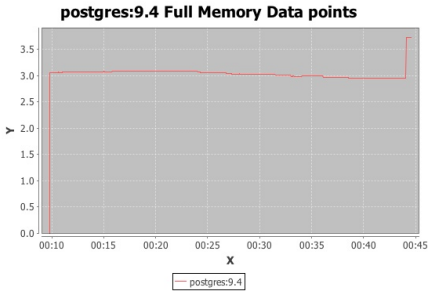
(a) Result service container CPU stats



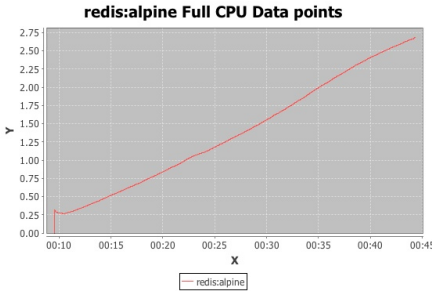
(b) Vote service container CPU stats



(c) Worker service container CPU stats



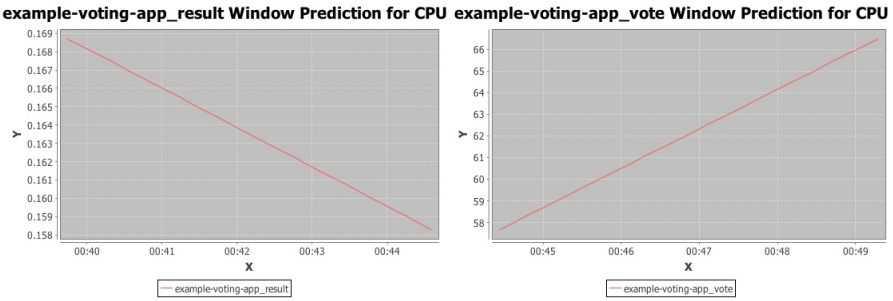
(d) db service container CPU stats



(e) Redis service container CPU stats

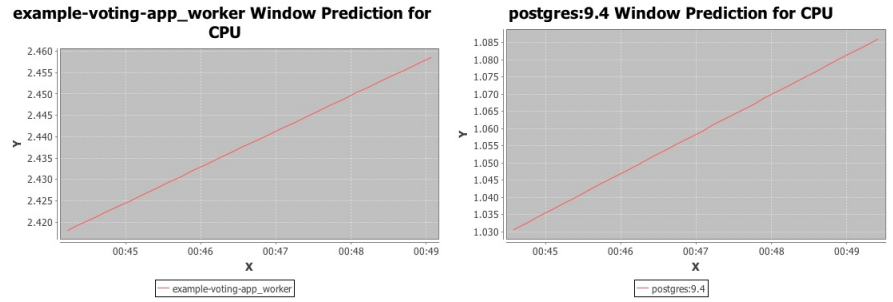
Figure 5.2: Plots of the monitored CPU statistics from the containers of the example voting application.

These Images show the a small sample of the testing done on the application and after the confirmation of the basic communications between the Monitor and sensor components, I then moved to the Analysis Component. From this component, my aim was to see it perform a prediction of the usage for the next time window (5 minutes) given the analysis of the already recorded data. To visualise the results of this functionality, I used the same plotting mechanism implemented for the statistics and the prediction data can be seen in Figure 5.3, where the prediction of the CPU data for the next time window is performed based on the data from the results in Figure 5.2.



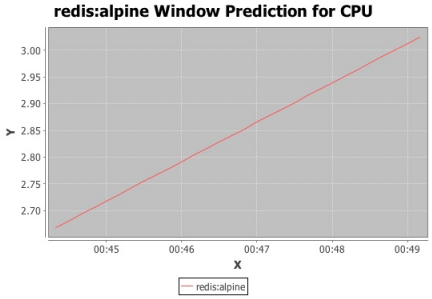
(a) Result service container CPU prediction stats

(b) Vote service container CPU prediction stats



(c) Worker service container CPU prediction stats

(d) db service container CPU prediction stats



(e) Redis service container CPU prediction stats

Figure 5.3: Plots of the predicted CPU statistics from the containers of the example voting application.

5.3 FULL SYSTEM TESTING

After the confirmation of the Monitor and analysis components, the next step of testing brought us to the version of the system after the implementation of the fuzzy logic functionality in the Analysis component, which is meant to use the predicted data and decide how many containers need to be added to or removed from the service, and the Plan component, which works with that data/ recommendation from the Analysis component and proposes the topology that best suits the recommendation before notifying the execution component to change the topology.

In order to confirm the above, I had to run varying loads through the application so that the system would respond accordingly. So, the Locust load simulation tool was used for this purpose and the work loads to be tested with are presented in Table 5.2 below. Table 5.1 shows the simulated topology structures of the test application Example Voting app introduced in the Test suite section these topologies were used for the purpose of providing the planning component a simplified example of different topology structures that it can switch between. The column show the number of topologies in each of the services for the given topology.

Vote	Result	db	redis	worker
1	1	1	1	1
2	1	1	1	1
3	2	1	1	1
4	3	2	1	1
5	4	3	2	1
6	5	4	3	2

Table 5.1: Topology options for the testing of the application

For the first test case, I ran different fixed numbers of users but this didn't cause a reaction from my system as seen in Table 5.2. So, I decided to use a different strategy of testing the application, which is presented in Table 5.3

Test	Number of users	Requests (s)	average response time	Result	Topology selected
1	30	2.9	11(ms)	No Adaptation	N/A
2	5	0.4	12 (ms)	No Adaptation	N/A
3	120	12	12 (ms)	No Adaptation	N/A
4	300	30	14 (ms)	No Adaptation	N/A
4	500	51	15(ms)	No Adaptation	N/A

Table 5.2: Test case 1 run on the application.

For this test case, I put a final user count of 10000 users and made the locust load tester hatch the users at a different rate every time the analysis was run (every after a time window). The results of this testing are as seen below.

Number of users	Requests(s)	average response time(ms)	Result	Topology selected
290	30	17	No Adaptation	N/A
540	55	15	No Adaptation	N/A
830	84	19	No Adaptation	N/A
1135	115.5	37	No Adaptation	N/A
1433	130	175	No Adaptation	N/A
2056	200	2370	No Adaptation	N/A

Table 5.3: Test case 2 run on the application.

As seen from the above tests in Table 5.3, There was still no reaction from the automation system's Planner and analysis so, I made some changes to the program performed the final testing phase, which is presented in Table 5.4. The following tests proved that the components were communicating successfully and the call for adaptation was possible and happening. So, the next step from here was to move the system to an application which I picked to perform the optimisation on as a case study, which is introduced in the next section.

Number of users	Requests(s)	response time (ms)	Result	Topology selected
300			Adaptation	2
400			No Adaptation	N/A
1150			No Adaptation	N/A

Table 5.4: Test case 3 run on the application.

6 EVALUATION AND CONCLUSION

The objective of this work was to design, develop, and test an IDE supporting the lifecycle seen in the paper [3]. To evaluate the system developed, I shall present a checklist of requirements that were realised in Chapter 3 and show the ones fulfilled by the system. I will then look at the functionalities not fulfilled, some of the issues left unresolved and the effect on the complete functionality of the system. Finally, some of the future work in the field and for this particular system is presented and then, I shall conclude this report.

6.1 EVALUATION

In this section, I review the requirements realised by the developed system using the tables below containing the requirement and whether it was implemented into the system or not and a relevant test that corresponds this functionality.

System Requirements

Requirements	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11
Realised	✓	✓	✓		✓	✓	✓		✓	✓	✓
Test											

Table 6.1: System Functional Requirements evaluation

Monitor Component Requirements

Requirements	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18
Realised	✓	✓	✓		✓			✓
Test								

Table 6.2: Monitor Component Functional Requirements evaluation

Analysis Component Requirements

Requirements	FR21	FR22	FR23	FR24	FR25	FR26	FR27	FR28	FR29	FR210	FR211	FR212
Realised	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Test												

Table 6.3: Analysis Component Functional Requirements evaluation

Plan Component Requirements

Requirements	FR31	FR32	FR33	FR34	FR35	FR36	FR37	FR38	FR39	FR310
Realised	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Test										

Table 6.4: Plan Component Functional Requirements evaluation

Execution Component Requirements

Requirements	FR41	FR42	FR43	FR44	FR45	FR46	FR47	FR48	FR49	FR410	FR411
Realised	✓		✓	✓	✓	✓	✓			✓	
Test											

Table 6.5: Execution Component Functional Requirements evaluation

6.2 UNRESOLVED ISSUES

* Reactive Analysis
* Implementing and using a Knowledge base for historical data analysis
* Providing other API options to be able to monitor non docker applications and other cloud applications.
* Unfulfilled requirements.
* Future Work

6.3 CONCLUSION

Summary

APPENDIX

B

APPENDIX

BIBLIOGRAPHY

[1] *AWS Auto Scaling*. https://aws.amazon.com/autoscaling/. Accessed: 2018-08-15.

[2] Fahd Al-Haidari, M Sqalli, and Khaled Salah. "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources." In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 2. IEEE. 2013, pp. 256–261.

[3] Vasilios Andrikopoulos. "Engineering Cloud-based Applications: Towards an Application Lifecycle." In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2017, pp. 57–72.

[4] Vasilios Andrikopoulos, Santiago Gómez Sáez, Frank Leymann, and Johannes Wettinger. "Optimal distribution of applications in the cloud." In: *International Conference on Advanced Information Systems Engineering*. Springer. 2014, pp. 75–90.

[5] Danilo Ardagna, Elisabetta Di Nitto, Giuliano Casale, Dana Petcu, Parastoo Mohagheghi, Sébastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, et al. "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds." In: *Proceedings of the 4th International Workshop on Modeling in Software Engineering*. IEEE Press. 2012, pp. 50–56.

[6] Maricela-Georgiana Avram. "Advantages and challenges of adopting cloud computing from an enterprise perspective." In: *Procedia Technology* 12 (2014), pp. 529–534.

[7] Tobias Binz, Gerd Breiter, Frank Leyman, and Thomas Spatzier. "Portable cloud services using tosca." In: *IEEE Internet Computing* 16.3 (2012), pp. 80–85.

[8] Pablo Cingolani and Jesús Alcalá-Fdez. "jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming." In: *International Journal of Computational Intelligence Systems* 6.sup1 (2013), pp. 61–75.

[9] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. "Elastic virtual machine for fine-grained cloud resource provisioning." In: *Global Trends in Computing and Communication Systems*. Springer, 2012, pp. 11–25.

[10] *Develop with Docker Engine SDKs and API*. https://docs.docker.com/develop/sdk/. Accessed: 2018-08-23.

[11] *Developing and Testing Microservices with Docker*. https://github.com/mjhea0/node-docker-api. Accessed: 2018-08-25.

[12] *Docker Compose*. https://docs.docker.com/compose/. Accessed: 2018-08-23.

[13] *Example Voting App*. https://github.com/dockersamples/example-voting-app. Accessed: 2018-08-25.

[14] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. "RPPS: a novel resource prediction and provisioning scheme in cloud data center." In: *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE. 2012, pp. 609–616.

[15] Stefan Frey, Claudia Lüthje, Christoph Reich, and Nathan Clarke. "Cloud QoS scaling by fuzzy logic." In: *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE. 2014, pp. 343–348.

[16] José María García, Octavio Martín-Díaz, Pablo Fernandez, Antonio Ruiz-Cortés, and Miguel Toro. "Automated analysis of cloud offerings for optimal service provisioning." In: *International Conference on Service-Oriented Computing*. Springer. 2017, pp. 331–339.

[17] Mostafa Ghobaei-Arani, Sam Jabbehdari, and Mohammad Ali Pourmina. "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach." In: *Future Generation Computer Systems* 78 (2018), pp. 191–210.

[18] Santiago Gómez Sáez, Vasilios Andrikopoulos, Florian Wessling, Clarissa Cassales Marquezan, et al. "Cloud Adaptation & Application (Re-) Distribution: Bridging the two Perspectives." In: (2014).

[19] Didac Gil De La Iglesia and Danny Weyns. "MAPE-K formal templates to rigorously design behaviors for self-adaptive systems." In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10.3 (2015), p. 15.

[20] Jeffrey O Kephart and David M Chess. "The vision of autonomic computing." In: *Computer* 36.1 (2003), pp. 41–50.

[21] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. "Automated control for elastic storage." In: *Proceedings of the 7th international conference on Autonomic computing*. ACM. 2010, pp. 1–10.

[22] *Locust*. https://docs.locust.io/en/stable/index.html. Accessed: 2018-08-25.

[23] Joao Loff and Joao Garcia. "Vadara: Predictive elasticity for cloud applications." In: *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*. IEEE. 2014, pp. 541–546.

[24] *Microservices*. https://martinfowler.com/articles/microservices.html. Accessed: 2018-08-04.

[25] *Oracle Solaris Containers*. http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html. Accessed: 2018-08-23.

[26] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. "Cloud container technologies: a state-of-the-art review." In: *IEEE Transactions on Cloud Computing* (2017).

[27] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. "Auto-scaling web applications in clouds: A taxonomy and survey." In: *ACM Computing Surveys (CSUR)* 51.4 (2018), p. 73.

[28] Alfonso Quarati, Daniele D'Agostino, Antonella Galizia, Matteo Mangini, and Andrea Clematis. "Delivering cloud services with QoS requirements: an opportunity for ICT SMEs." In: *International Conference on Grid Economics and Business Models*. Springer. 2012, pp. 197–211.

[29] *RKT overview page*. https://coreos.com/rkt/. Accessed: 2018-08-23.

[30] Lenar Yazdanov and Christof Fetzer. "Vertical scaling for prioritized vms provisioning." In: *Cloud and Green Computing (CGC), 2012 Second International Conference on*. IEEE. 2012, pp. 118–125.

[31] Lotfi A Zadeh. "Outline of a new approach to the analysis of complex systems and decision processes." In: *IEEE Transactions on systems, Man, and Cybernetics* 1 (1973), pp. 28–44.

[32] Lotfi A Zadeh. "Fuzzy logic= computing with words." In: *IEEE transactions on fuzzy systems* 4.2 (1996), pp. 103–111.

DECLARATION

Put your declaration here.

Netherlands, May 2018

Eric Rwemigabo