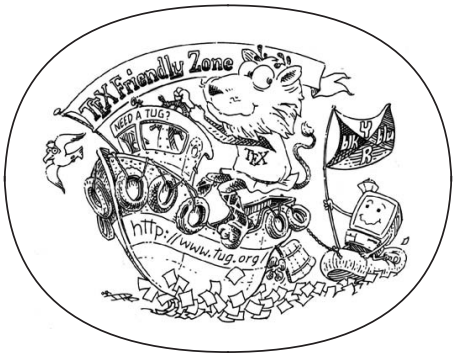
ERIC RWEMIGABO

AUTONOMIC APPLICATION TOPOLOGY REDISTRIBUTION

[August 22, 2018 at 14:30 – classicthesis version 1.0]

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

May 2018

[August 22, 2018 at 14:30 – classicthesis version 1.0]

Eric Rwemigabo: *Autonomic application topology redistribution,* Autonomic Computing, MSc Computing Science (Software Engineering and Distributed Systems), © May 2018

SUPERVISORS:
Vasilios Andrikopoulos
Mircea Lungu

LOCATION:
Netherlands

TIME FRAME:
December 2017 – May 2018

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

Dedicated to the loving memory of Abdul-Malik N E Umaru.
1992 – 2017

ABSTRACT

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— knuth:1974 [knuth:1974]

ACKNOWLEDGMENTS

Put your acknowledgments here.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

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 [21]. 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 [16] 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 [12] 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 [18]), 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 [20], 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 [20] 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 [16] 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 [19] 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 [24] points out that the fuzzy logic concept stems from the Computing with Words methodology, which all started from [23]. 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 [11], 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 a later chapter.

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 [10, 13, 22] 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 [20]. 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 [11].

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[20]. The solution of the use of dynamic parameters as seen for example in [17] 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 [14] 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 order 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.

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 this estimation 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 in the Knowledge base.

FR33 The Plan Component should have access to the suggested topology from the Analysis Component in the Knowledge base.

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 baby 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 in the Knowledge base.

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

FR37 The Plan Component should have record of the currently running or deployed topology.

FR38 The Plan Component should notify the Execution component if a change is required.

FR39 The Plan Component should store the new topology for the (re-)distribution in 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 in the Knowledge base.

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 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.

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 use prediction is outside the SLO.

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

11a No 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).

3.2.2 *Design patterns*

3.3 COMPLETE SYSTEM ARCHITECTURE

3.3.1 *Component Diagram And Patterns*

- Logical view of system

4 IMPLEMENTATION

For this chapter, the details of the implementation of the system are presented including a Component Diagram showing the particular components of the system and how they interact and share data, some of the design patterns used, some of the technologies that were helpful and necessary to the implementation of some of the system's components including their use to the system and finally, a deeper look into the particular components that make up the system and their functionalities.

4.1 TECHNOLOGIES AND THEIR UTILISATION IN THE SYSTEM

4.1.1 *Java*

4.1.2 *Docker*

NB: Show ease of use / microservice compatibility

4.1.2.1 *Docker API*

4.1.2.2 *Docker Compose*

4.1.3 *MySQL*

4.1.4 *Fuzzy Logic*

4.2 APPLICATION COMPONENTS

[15]

4.2.1 *Monitor*

4.2.2 *Analyse*

4.2.3 *Plan*

4.2.4 *Execute*

4.2.5 *Knowledge base*

4.3 ALGORITHMS AND DATA STRUCTURE??

5 SYSTEM TESTING AND EVALUATION

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 completing most of the implementation of the system's components, that is until the level of the Analysis of the data that is collected by the system, 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 these tests.

5.1 MONITOR AND ANALYSIS COMPONENT TESTING.

5.2 PLAN AND EXECUTION TESTING.

5.3 FULL SYSTEM TESTING AND EVALUATION.

6 CONCLUSIONS

Part I

APPENDIX

A

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] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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