National College of Ireland

Docker Container Reactive Scalability and Prediction of CPU Utilization Based on Proactive Modelling

MSc Research Project
Cloud Computing

Aravind Samy Shanmugam
x16104421

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet – 2016/2017
School of Computing

Student Name:	Aravind Samy Shanmugam
Student ID:	x16104421
Programme:	Cloud Computing
Year:	2016
Module:	MSc Research Project
Lecturer:	Dr. Catherine Mulwa
Submission Due Date:	16/08/2017
Project Title:	Docker Container Reactive Scalability and Prediction of CPU Utilization Based on Proactive Modelling
Word Count:	XXX

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	13th September 2017

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Docker Container Reactive Scalability and Prediction of CPU Utilization Based on Proactive Modelling

Aravind Samy Shanmugam
x16104421
MSc Research Project in Cloud Computing

13th September 2017

Abstract

The resource utilization of cloud-based applications has been used for capacity planning and forecasting resource demand. Cloud scalability provides significant scaling techniques to identify real-time demand as well the future demand to reduce resource wastage and network bandwidth. The project focuses to tackle the Docker container reactive scalability and prediction of CPU utilization using proactive modelling. The docker container reactive scaling is structured based on threshold policies on cloud and candidate performed scaling in and scaling out of docker container web services built on CPU utilization threshold. The CPU metrics is perceived on cadvisor dashboard where HAproxy is utilized to distribute load to container web services based on round-robin algorithm. In addition, proactive scaling is performed using ARIMA time series analysis model. The public cloud Amazon Web Services is used to perform container reactive scaling and containerized open source jupyter notebook application is utilized to perform prediction. Based on CPU utilization data observed on amazon cloud, the proactive ARIMA model is trained. The parameter combination values are evaluated using Akaike Information Criterion value to get best results for forecasting CPU utilization. The paper explores the evaluation results of container reactive scaling and ARIMA prediction CPU utilization. In addition, the state of the art results are also demonstrated.

1 Introduction

Cloud computing technology is widely used in many organizations that provides highly available IT resources. The introduction of Docker containers provides numerous advantages to industries in many aspects. The lightweight containers are very efficient and it can deploy easily on any platforms with significant less overhead than existing techniques. The scalability in cloud plays a vital role, which manages the application seamlessly and deliver the required resources based on its demand. The scaling techniques such as reactive and proactive on docker containers springs the high availability for the application to users.

1.1 Motivation and Background

The capacity planning in cloud provides a platform for resource management and effective resource utilization. The cloud consumers are facilitated with pay-as-you-go model where clients only pay for the resource usage. The capital expenditure in several organizations are tremendously condensed with the deployment of cloud and docker containers. The operational expenditure is comparatively low because of the reduction in resource wastage, network bandwidth which reflects the low latency to end users. The reactive and proactive scaling aids the application to work continuously with less downtime. The reactive scaling provides resource on real-time demand and proactive scaling estimates the future demand of the application that is facilitated on monitoring resource metrics such as CPU utilization and memory usage. The existing technique is adopted mostly based on reactive scaling and few in proactive scaling, still scalability on docker containers can be significantly improved. The project approach comprises of container reactive scaling based on threshold limit and proactive time series analysis ARIMA model to predict and forecast the CPU usage. To the best of my knowledge, the results of this project will be significant and vital to the cloud computing community. The project is conducted on public cloud Amazon Web Services. The candidate uses base image Ubuntu 16.04, docker container and scalability techniques based on CPU resource utilization.

1.2 Project Specification

This project tackles the issue of docker container scalability. In addition, the paper proposes the implementation and evaluation of reactive scaling and prediction of CPU utilization based on proactive modelling. To tackle the proposed solution, the below research question is specified.

1.2.1 Research Question

Currently there exists problem on determining equal balance of load when container web service scaling takes place. *RQ-" Can CPU load stress on docker containers (web services) scale and balance the load using HAproxy load balancer based on reactive threshold limit technique?"*

Use Case Scenario: "If any container limit reaches threshold CPU utilization, ***scale out of container web services*** *takes place otherwise,* ***scale In of container web services*** *takes place."*

Sub RQ- "Can the proactive ARIMA time series analysis model predict and forecast CPU utilization?"

1.2.2 Research Objectives

To address the research question: Objective 1, investigation of CPU resource utilization using reactive and proactive scaling models on Docker container. Objective 2, tackles the design, implementation and evaluation of docker container reactive scaling. Obj 2(i), design and implementation of docker container reactive scaling based on threshold-policies [shown in section 4.2, 4.3] Obj 2(ii), results and evaluation of docker container reactive scaling (scaling out and scaling in of containers based on CPU threshold) [shown in section 5.2.1] Objective 3, tackles the implementation of proactive ARIMA model of CPU data observed on AWS using containerized jupyter notebook [shown in section

4.4] evaluation and prediction results of CPU usage [shown in section 5.2.2] Obj 3(i), visualization of CPU resource utilization data before implementing the prediction model. Obj 3(ii), implementation and evaluation of CPU utilization data using ARIMA model. Obj 3(iii), ARIMA one-step further forecast prediction. Obj 3(iv), forecasting CPU utilization. Objective 4, documentation of the NCI technical report and configuration manual

The specified objectives are mandatory to address the research question. To juggle these objectives, a comprehensive review is conducted.

1.2.3 Thesis Contribution

This project contributions the following to the body of knowledge: (i) Results of reviewed literature on docker container reactive scalability and prediction of CPU utilization from 2014-2017 (ii) Computations of the docker container reactive scalability and fully evaluated results from this computations (iii) A fully developed and evaluated proactive ARIMA prediction model of CPU usage (iv) Results of scaling out and scaling in of docker containers based on CPU threshold and visual presentation of proactive model results (v) A technical technical report and configuration manual which have been submitted to NCI as a requirement for the MSc.

The rest of this report is structured as follows. Section 2 presents a critical review of CPU resource utilization using reactive and proactive scaling models. Based on the results of the comprehensive review and the identified gaps in the body of knowledge, in Section 3 depicts the methodology of proposed approach on container reactive scalability and prediction. Section 4 tackles the design and implementation of docker container reactive scalability and proactive ARIMA model prediction of CPU usage. In Section 5 depicts the evaluation and results of docker container reactive scalability and ARIMA prediction of CPU usage. Finally, Section 6 illustrates the conclusion of the project and provides an enhancement for the future work.

2 Related Work

This section presents a review of reactive and proactive scaling models on docker containers using CPU utilization . More specifically it focuses on CPU Utilization on Reactive and Proactive Models developed from 2014 to 2017. Furthermore investigation of mitigation on docker container scalability and predictions is conducted. Finally a critical review of the ARIMA prediction model is conducted and identified gaps presented.

2.1 CPU Utilization on Reactive and Proactive Models

Cloud elasticity has remained a platform that offers essential resources in line with varying workload demands. Kan (2016) established DoCloud gears container elasticity corresponding to disparity in web application resource. It entails reactive and proactive model to tackle horizontal scalability. The experiment is unified with ARMA proactive model and threshold-built reactive model. The scalability algorithm encompasses CPU utilization and memory usage of Docker containers that is gauged using HAproxy proceeding FIFA 1998 world cup traces and randomly simulated workloads. The architecture is not inspected with whole log files and its constrained scale in, scale out results deliver additional scope on scaling prediction on containers.

Computing applications resource utilization plays a vivacious role in terms of scalability prediction thereby resulting in reduced resource wastage and saving huge expenditure. Meng et al. (2016) offered CRUPA, that applies resource to forecast container scalability. The proposed model is executed using ARIMA time series analysis model. The algorithm involves CPU utilization that adapts horizontal reactive scalability. The CRUPA is deployed on Docker container to encounter predictive container assessment. The experimental scope is extended in ameliorating the prediction accuracy using load balancer on web application facilitating investigation on container scalability to reduce resource wastage.

The energy efficiency and performance of data centres isn't quite remarkable in capacity planning. Alzahrani et al. (2016) considered SLA and developed Energy-Built Auto Scaling (EBAS) method encompassing dynamic voltage frequency scaling technique. It proactively predicts the scalability on containers by using CPU utilization. It assumes ARIMA model for scaling prediction of web application. The demonstrated results in effective resource provisioning with improved energy efficiency. The research has supplementary scope in forecasting accuracy involving parameter decision of time series model and evaluation grounded on large scale applications.

2.2 Identified Gaps in Technologies Used

The scalability and prediction of CPU resource utilization by means of docker containers is recognized as research gap. The research work acknowledges the exiting work and can be significantly upgraded with reactive and proactive scaling model in terms of effective docker environment intended for reactive scaling and parameter decision designed for better prediction.

2.3 Mitigation on Docker Container Scalability and Prediction

The container scalability is offered using resource utilization, for example response time of web requests created on PAS algorithm includes proportional-integral-derivative controller (PID). de Abranches and Solis (2016) estimated the experiment with numerous workloads using Docker container, HAproxy load balancer and Redis database. The research scope classifies the importance of improving the PID parameters to scale containers. The implementation is advanced and built on control theory scalability technique that eventually lacking the effective comparison to prove better results on elasticity through existing techniques.

The Docker container performance resource utilization is evaluated and documented on many aspects to tackle better lightweight tool for effective application sharing. N et al. (2015) evaluated container performance with Bonnie++ benchmarking tool and remaining metrics such as CPU utilization and memory usage are evaluated with psutil benchmarking code. The evaluation is shown between Docker containers and host machine on disk usage, memory usage, network I/O. As a result, containers outperformed host system on resource utilization. The performance comparison provides better results yet the research scope is focused on enhancing Docker scalability and security. Moreover, no further technique was adopted and implementation was not shown to extend the support aimed at evaluation.

Baresi et al. (2016) planned MicroCloud architecture that provides effective resource management built on many containerized applications. The framework is built on ECo-

Ware describes the self-resource adaptation that uses meta-workflow technique to tackle applications. It consists of TOSCA library to facilitate the infrastructure and it is demonstrated on Amazon EC2 to evaluate the resource metrics like average response time. Although the container built resource management provides enhanced results, the performance metric comparison was not carried out for evaluation. The scalability technique was not accepted and research remained intended to deploy the same architecture on big data applications.

The advancement in container application deployment facilitates the microservice architecture. Inagaki et al. (2016) defines Docker container management that offers insight on core and container scalability. It identifies the bottlenecks of respective core and container scalability that affects the performance of container management. This hierarchical approach mitigates the bottleneck in operations by analysing multiple layers at Docker daemon. The results aid to improvise Docker container scalability and elasticities further research scope scheduled emerging microservice applications.

Hegde et al. (2016) investigates provisioning of large scale application on containers in addition identified bottlenecks in the aspects of scaling factors and system parameters. To overcome operating system process overhead, SCoPe is developed in order to provide decision on application partitioning and provisioning method to tackle the number of containers required for every application to run parallel on host machine. The proposed model is executed and evaluated its performance using real data set and documented the impact of SLA with existing approaches. Although, the SCoPe produced better results, the decision system has not been structured built on resource utilization of the system. The provisioning decision system bottlenecks and future work is recognised that offers the way to work on system utilization rate and execution of real time workloads on containerized applications.

The integration of container and video surveillance analysis technique has overcome the bottleneck that exists on virtual machine. Zhang et al. (2016) proposed container built proactive resource provisioning for video microservices. The prediction is built on nearest neighbour regression time series model that predicts future resource demand in addition to usage Docker container to deliver video microservices. The results are evaluated with other prediction models like ARIMA and DRFA and affords better accuracy and higher deployment density. The model can be significantly predicted in better way without constrained observation and effective parameter decision on other models.

Kim et al. (2017) proposed container traffic analyser to increase network traffic and minimize horizontal scaling out time. LTLB load balancing algorithm is developed on the basis of collection of real-time network traffic information. The algorithm is implemented by means of Docker container on AWS and evaluated with virtual machine. Auto scaling mechanism is deployed also proactive scaling built on traffic and resource usage is proposed and compared with reactive scaling. The future research scope stated on deploying the proposed scaling technique in order to register low latency and less cloud resource wastage.

The microservices are widely used to deliver specific functions and it is effectively deployed on containers. Moradi et al. (2017) shown ConMon, a monitoring system on containers which automatically monitors network performance whenever containers communicate each other. The ConMons monitoring controller does passive and active monitoring on containers. The experiment is conducted on Docker containers with Open vSwitch. The impact on resource metric along with application performance, traffic and scalability are graphically shown. The bottleneck for future research is identified as

applications throughput and latency was affected and cost can be resolved on passive monitoring on multiple application containers.

2.4 Critical Review of ARIMA Model

The cloud application providers need to sustain the competitive advantage to retain their customers with sufficient QoS. To overcome the problems and results of insufficient QoS, Calheiros et al. (2015) described that one of the major key factor affecting Quality of Service (QoS) is the dynamic workload which leads to variable resource demands. In order to circumvent the real-time workload exceeds the resource capacity problem, a proactive approach is structured using the Auto Regressive Integrated Moving Average (ARIMA) model. The approach uses workload analyser component and feeds the result with accurate predictions that enables rest of the system to scale the resources without any wastage. The impact of ARIMA model accuracy in terms of efficiency in resource utilization and QoS was also evaluated. The simulation results showed an accuracy of up to 91 percent which obviously resulted in efficiency of resource utilization with minimal impact in response time of users. The research scope is planned to integrate robust techniques for forecasting peak resource utilization and investigates on load prediction modelling based on user constraints.

The paramount factor of cloud services is to ensure quality of experience (QoE) to end users. Dhib et al. (2016) used Massively Multiplayers Online Gaming (MMOG) traces to predict future workload demand. Seasonal Autoregressive Integrated Moving Average (SARIMA) model was proposed to forecast workload behaviour. The algorithm is implemented to predict the resource allocation built on the forecasted result of SARIMA. The experiment is evaluated which address the impact of QoE by proposed algorithm. The documented results provided accuracy on normal behaviour and future research scope stated to test the algorithm with various workload scenarios especially during unexpected behaviour phase. The proactive resource provisioning method was exploited where forecasting accuracy can be improved significantly on perceiving many MMOG users simultaneously.

The cloud application resource demand varies in accordance with consumer requirement. The prediction of cloud resource needs always secure a significant position to estimate the future workloads. Erradi and Kholidy (2016) proposed DDHPA, a hybrid prediction approach which integrates two methods MSVR and ARIMA to predict resource utilizations in terms of CPU, memory with higher accuracy. Other metrics such as response time and throughput can be predicted to decide on resource scalability. The DDHPA results outperform existing prediction models in terms of CPU utilization which is evaluated by MAPE and RMSE. In future, the research scope is to test the model with vast dataset in a real-time environment.

The cloud elasticity aids in capacity planning which facilitates the scaling of cloud resources. Hu et al. (2016) encountered scaling the virtual machine consumes more time which results in inefficiency and apparently affects QoS and latency. The proposed prediction core modules such as monitor and predictor identifies the existing problem and helps to predict the resource built on workload demand well in advance. The predictor such as ARIMA, SVM was used to evaluate the metrics like under provisioning and over provisioning of resource. The framework enhances the latency and cloud QoS and re search scope stated the model will be improved constructed on users demand and resource metrics.

The reactive scaling provision in cloud infrastructure is not determined to identify the resource demand of an application. Balaji et al. (2014) conducted comparative study on two prediction models Holt-Winter and ARIMA using public dataset to monitor resource usage. The experiment results provided Holt-Winter performed better than ARIMA and it is evaluated with some metrics such as MAPE and RSME. Since the evaluation is conducted on constrained data thus follows similar pattern, it can be measured significantly on large data set with improvised parameter decision. The comparative study stated further research scope on different data which includes metrics such as throughput and resource utilization.

2.5 Conclusion

Based on the reviewed literature and the identified gaps, the project is proposed to perform docker container reactive scaling and prediction of CPU resource utilization by means of proactive ARIMA model.

3 Methodology Approach Used

This section specifies the iterative software methodology which was used. The iterative model is a particular implementation of a software development life cycle (SDLC) that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final product is complete.

3.1 Proposed Research Approach

The project method is proposed to scale docker containers based on reactive threshold limit and predict CPU utilization by means of proactive ARIMA model.

3.1.1 Software Development Methodology Used

The Software development methodology castoff in this project is depicted in Figure 1. The project flow is designed as: (i) planning phase helps the initial planning of project on docker container scaling technique (ii) requirement phase entails system hardware and software requirements towards execution of the proposed plan (iii) design analysis phase encompasses docker container reactive scaling and trained proactive ARIMA model to forecast CPU utilization (iv) implementation and deployment phase pays the comprehensive proposed model implementation techniques and deployment (v) testing phase confer the test on reactive scaling and prediction of CPU usage by means of open data to receive results (vi) evaluation phase includes result evaluation of container scalability and prediction.

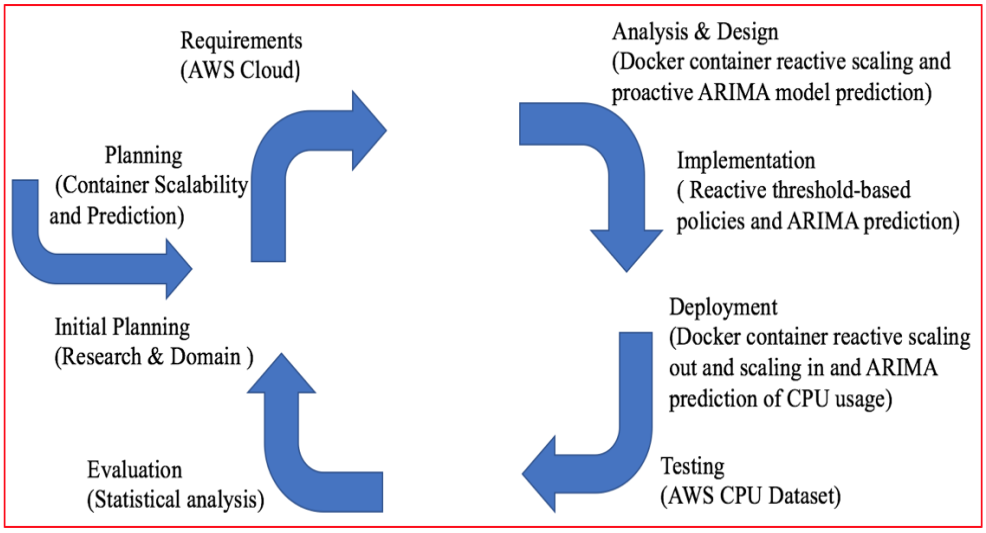


Figure 1: Software Iterative Model

3.1.2 Technologies and Softwares Used During Implementation

This section presents the tools and software technologies used in the implementation of the proposed solutions in Section 4.

(i) Docker: Docker is an open source software that provides containerized platform for agile deployment of applications besides it allows shipping of an application as package wherever with a smaller amount overhead. Meanwhile, it uses kernel directly. Docker containers are distributed by docker daemon which is the environment core. N et al. (2015) quantified Docker has major components, (i) Docker Image (ii) Docker container and (iii) Docker registries. Docker image makes Docker containers and it is available in repository. The container bounces directory location then offers environment aimed at any application to run. The docker architecture is represented in Figure 2.

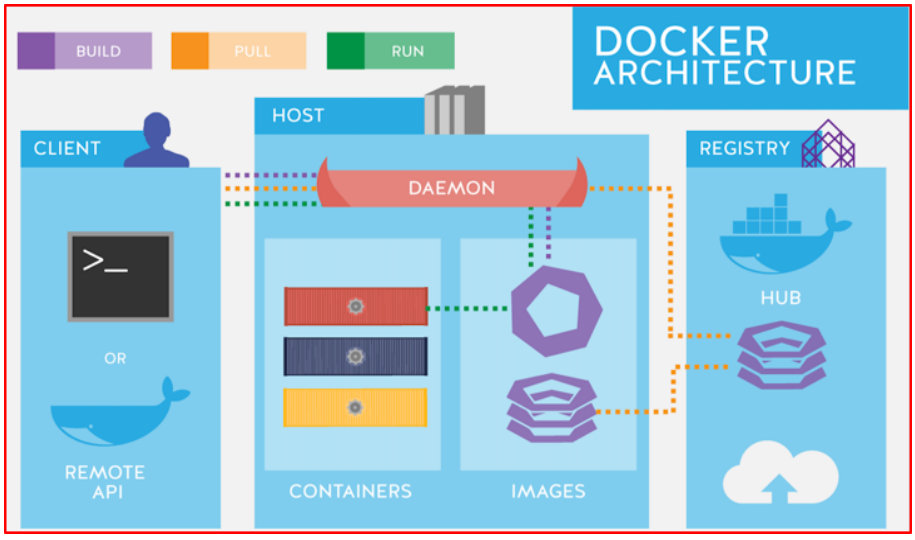


Figure 2: Docker Architecture

(ii) HAproxy: HAproxy is an open source software that denotes high proxy availability.

It organizes load balancing function to enhance the reliability of server. It works based on round robin algorithm in default and distributes workload and balance the network traffic among multiple servers. Based on users web request content, it uses IP range and traffic will be forwarded to backend server by HAproxy load balancer.

(iii) Scalability Techniques: Cloud scalability techniques are broadly classified as reactive and proactive scaling. Reactive scaling prepares the instance scaling in and scaling out in accordance to real-time demand of an application where threshold built policies originate under this category, in the meantime it uses resource metric to raise or remove instance resource. Proactive scaling estimates and conjecture the future workload demand of an application, while using resource metric to predict resource in terms of time series analysis of data and subsequently it allocates or deallocates instance depends upon the requirement. Other scaling techniques such as reinforcement learning, queuing theory and control theory arises either one of the two categories.

(iv) Threshold-based Policy: Built on resource utilization such as CPU and memory usage, this technique remains utilized in reactive approach and used by several top-notch cloud service providers. It adapts horizontal scaling anywhere, instance scaling out or scaling in takes places bases on threshold limit of CPU usage. Amazon EC2, Microsoft Azure and many cloud vendors uses threshold-based policy for instance scaling. Openjdk java platform is castoff to set CPU usage threshold limit.

(v) cAdvisor: Container advisor delivers on running container resource metric usage. It is facilitated to witness the performance of each container and provide statistical results on long term usage in terms of charts and histograms. It automatically ensures slight adjustment on container performance as well as predicts future usage.

(vi) Influxdb: Influxdb is an open source database on time series. It collects resource utilization and stores it in time series format for monitoring and performing operation on data analytics.

(vii) ARIMA Time series analysis model: ARIMA is an integration of autoregressive (AR) and moving average (MA) model with differencing (d) functions, it uses historical data to predict future value of the time series value. It incorporates autocorrelation function and partial autocorrelation function i.e. ACF and PACF. It has (p, d, q) parameters which illustrates p as order of AR, d as differencing operator and q as maximum order of MA. The prediction results depend on the parameter combination.

(viii)The Jupyter notebook: The Jupyter notebook is an interactive web application available as an open source that aids to create and deploy code in a flash. The IPython notebook offers an environment to work on python code and gives easy way to pictorially represent data results, simulation and many other purposes. The notebook dashboard provide access to local files. It contains mathematical code and equations that aids to perform data analysis in addition share documents easily.

4 Implementation of Docker Container Reactive Scaling and ARIMA Prediction of CPU Usage

This section presents the implementation and evaluation of the proposed solution. In order to solve the research question specified in Section 1.2.1 and address the problem on determining equal balance of load when container web service scaling takes place, the objectives specified in Section 1.2.2 are tackled and results presented.

4.1 Requirement Specification

The requirements specified for this project are depicted in Figure 3.

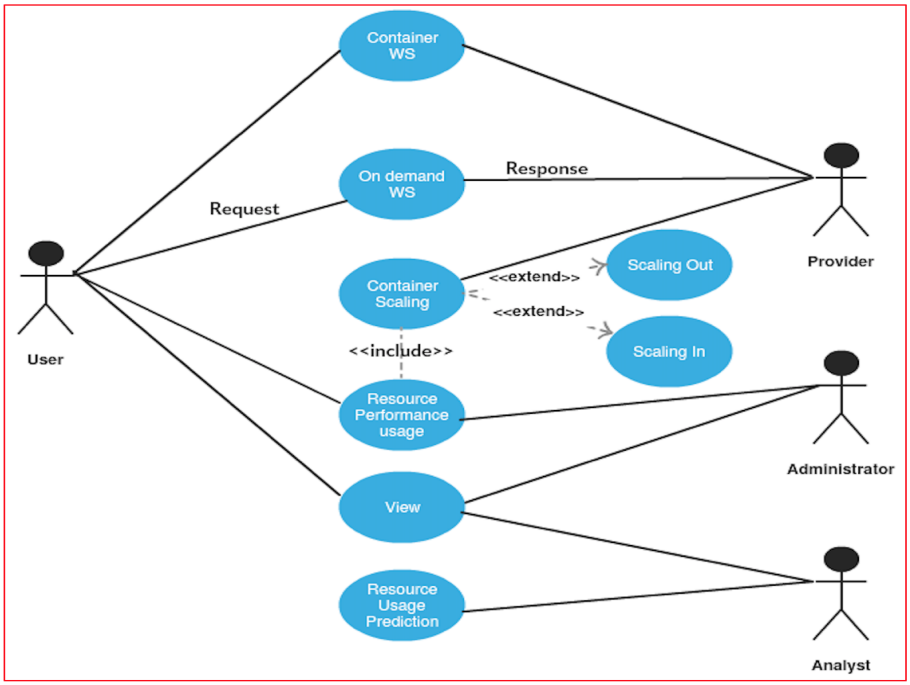


Figure 3: Container Scalability and Prediction Use case diagram

The UML use case diagram of container reactive scaling and prediction of resource usage is presented. The above diagram elaborates the container web service provision that can be accessed on the user end wherever it is facilitated through provider. The minute the workload request to web service hits on high proportion, the response will be provided on real-time demand somewhere container scaling initializes horizontal scaling out and scaling in technique based on CPU resource usage. The container CPU metric is monitored by administrator and provides visibility stats to user. Built on CPU resource usage, analyst can predict the future demand of web service CPU resource utilization scheduled using proactive time-series analysis model.

4.2 Architecture Design Specification

The project design and architecture specification is depicted in Figure 4, while it has three components namely presentation tier, business logic tier and data persistent tier. Each

tier involves of method and functionality to implement container scalability on docker and prediction built on CPU resource utilization.

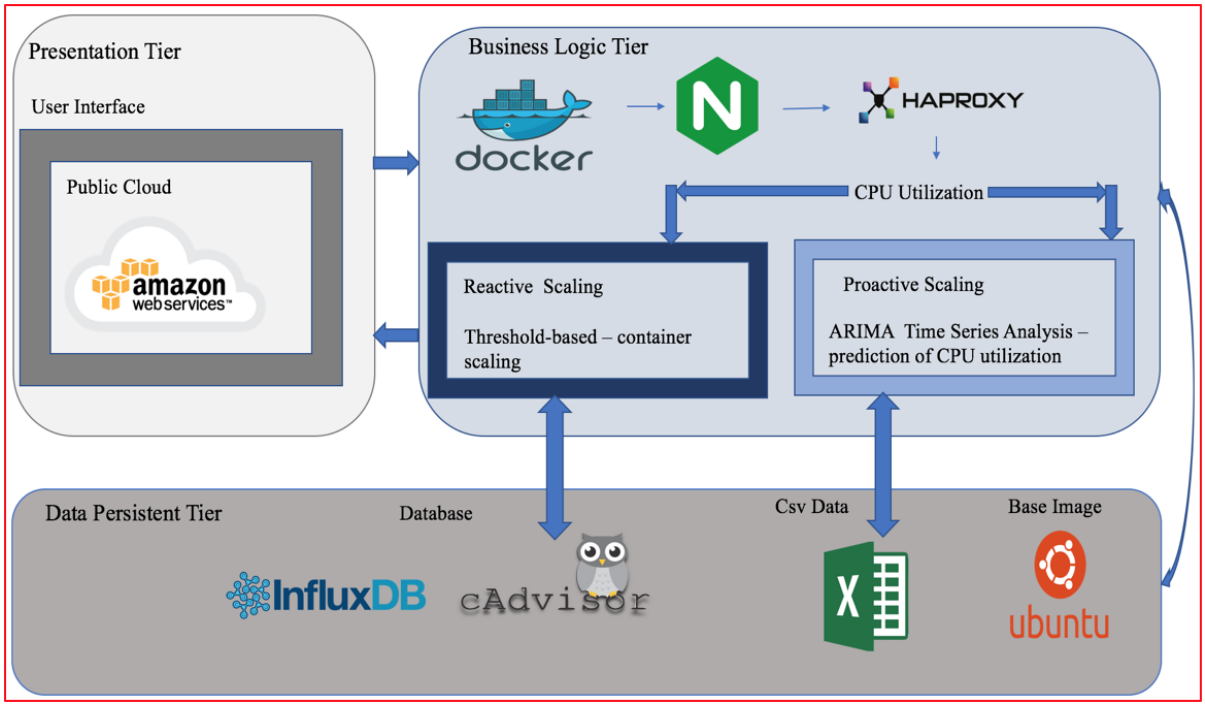


Figure 4: Architecture Design

(i)Presentation Tier: The presentation tier consists of user interface which provides an environment for implementation. The public cloud platform Amazon Web Services is used in this project for implementation. (ii) Business Logic Tier: The project encapsulates the two main objectives of implementation. The first main objective is on docker container scalability built on reactive threshold limit and the second main objective is to predict and forecast the CPU resource utilization of open data using proactive ARIMA time series analysis model. (iii) Data Persistent Tier: The workflow is monitored at the backend tier and it incorporates the functions of cadvisor and influxdb for monitoring CPU resource usage during reactive container scaling. The open CPU utilization csv data is analyzed for prediction and to perform forecasting trend of resource utilization.

4.3 Docker Container Reactive Scaling Implementation

This section presents the implementation of docker container reactive scaling and its work-flow is depicted in Figure 5.

(i) Each dockerfile is created made on docker images such as web service nginx, registrar, consul, HAproxy load balancer, reactive monitoring, cadvisor and influxdb. The docker images are successfully integrated with each other on composing together using docker command. It is significantly implemented using same ubuntu image which occupies less memory on cloud storage.

(ii) The nginx server web content application is deployed as containerized web service image as myws. The existing docker images are configured and the approach of reactive scaling regulates the function of docker container flow built on CPU resource threshold. In addition, we have configured and used registrar docker image to keep register and

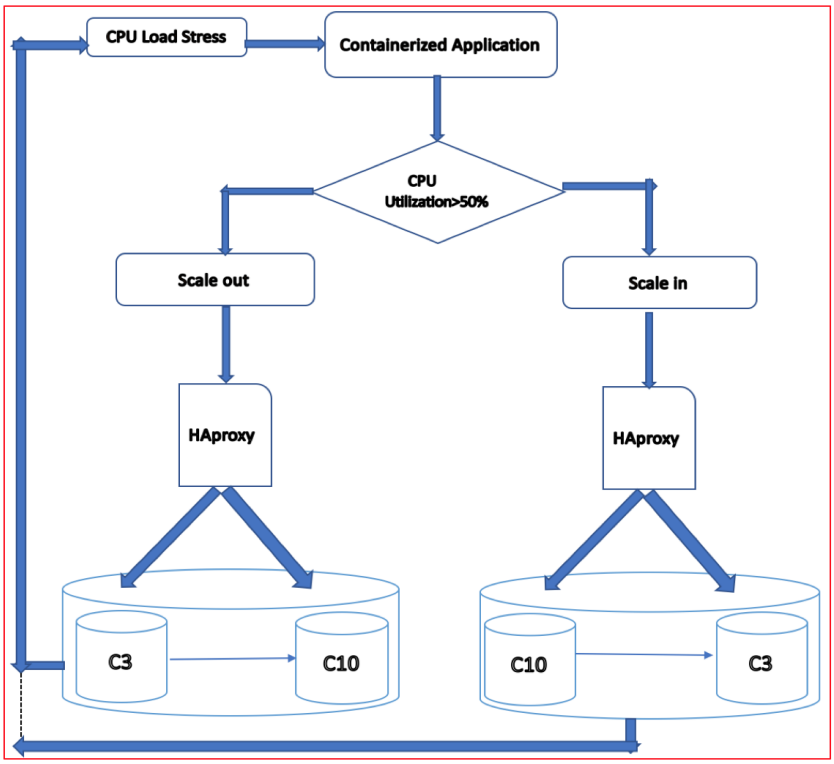


Figure 5: Workflow of Container Reactive Scaling

track on new container arrival and unregister on container removal. The information on container changes is identified and perceived by consul docker image. The registrar and consul work is incorporated with HAproxy load balancer. Built on consul template, load balancer initiates round robin algorithm and concurrently balance the traffic between the containers.

(iii) The reactive container scaling tackles the performance of containers built on CPU resource usage. The horizontal scale out, scale in concept is adapted in order to achieve container scalability. The container instance is initiated constructed on threshold limit. The candidate incorporated openjdk-8 open source java platform to implement threshold-built container scaling. When container web service is accessed and once CPU usage crosses above 50 percentage, at that time containers begin to scale up. Built on the concurrent load and simultaneous hit, the container reactive scaling out and scaling in technique is implemented.

(iv) Once the implementation of container threshold monitoring image is done, it is assigned to load balancer to evaluate scaling of containers. Built on CPU load stress, registrar and consul immediately acts and identifies the scaling process. The approach scales maximum of 10 containers and minimum of 2 containers. The container resource metric is configured and initiated simultaneously.

(v) The container reactive scaling is implemented while candidate tackles the performance of CPU usage and observed using cadvisor dashboard. The load balancing changes remain identified on HAproxy stats using extensive IP port range.

4.4 Prediction of CPU Usage using ARIMA Model

This section presents the implementation of proactive ARIMA model prediction of CPU usage and the workflow is depicted in Figure 6.

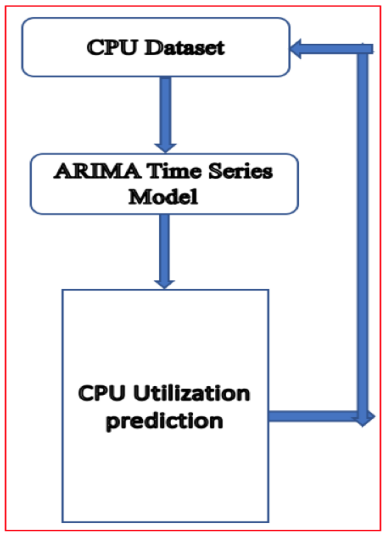


Figure 6: Workflow of Proactive ARIMA CPU Usage Prediction

(i) The analysis of CPU resource usage is thoroughly performed in order to predict and forecast the CPU utilization usage. The prediction analysis is conducted on public CPU usage data observed on AWS using proactive time-series approach. Anaconda 3-4.4.0 and containerized-scipy jupyter notebook is installed to before prediction.

(ii) The data cleaning is achieved to remove corrupted records. The mean value is calculated built on daily average of CPU usage and data filling is practical to fill missing values.

(iii) The ARIMA parameter value is decided made on grid search. It pledges the parameter combination with seasonal ARIMA to train and evaluate the model to get best results. In addition, stats models Akaike Information Criterion is used to automate the process. The candidate performed test on all possible results and selected the best data to predict forecast of CPU usage. The diagnostic analysis of data is shown graphically. Built on the result, seasonality and normality of CPU usage data is conducted for evaluation.

(iv) The prediction of CPU usage is initialized on defining date of prediction and dynamic function. It is performed to validate the prediction built on historical CPU usage data. Then one step further forecast and forthcoming months CPU usage forecasting is implemented built on prediction results.

4.5 Conclusion

The architecture design, implementation and work-flow technique of container reactive scaling and ARIMA prediction of CPU usage is shown that tackles the research objectives in section 1.2.2

5 Evaluation of Developed Solution

The evaluation is conducted in order to validate and test the docker container reactive scalability. In addition prediction ARIMA model is also evaluated and results presented.

5.1 Evaluation Process

The steps involved during the evaluations are depicted in Figure 7. During evaluation analysis, the following machine configurations are utilized. In addition to public cloud Amazon Web Service (AWS) platform. The specification of amazon EC2 instance is configured as Instance Type- t2.xlarge which has 4 vCPU, 16GB RAM, storage of 24GB and we have used base image of Ubuntu 16.04. To implement main objectives, docker version 17.06.0 ce is installed and a list of mandatory docker images are required to figure docker environment towards performance of reactive container scaling and containerized-scipy image is used. Anaconda-3-4.4.0-Linux version is utilized to install jupyter notebook intended for accomplishment of ARIMA prediction.

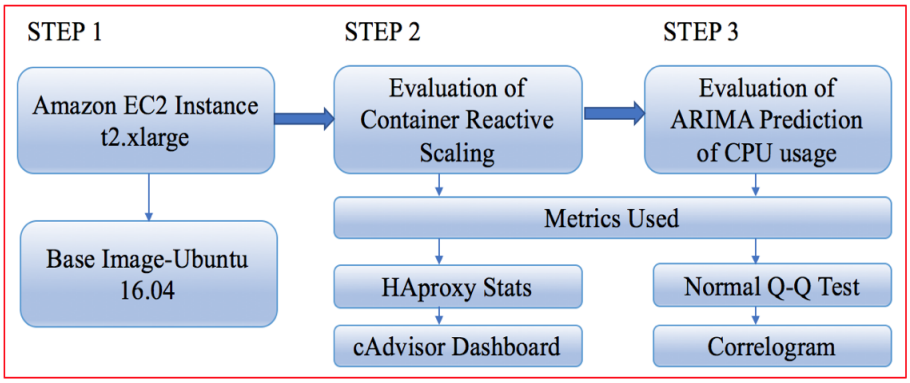


Figure 7: Evaluation Process

5.2 Evaluation and Results of Developed Solution

5.2.1 Evaluation and Results of Container Reactive Scaling

The docker container web service is initiated on successful deployment of all docker images. Initially the technique starts two web services with other HAproxy and cadvisor images. The container reactive scaling sets horizontal scaling concept grounded on CPU load. Figure 8 depicts the initialization of container reactive scaling images.

```
ubuntu@ip-172-31-7-248:~/dockerCompose$ sudo docker-compose up -d
Starting dockercompose_consul_1
Starting dockercompose_monitoring_1
Starting dockercompose_influxsrv_1
Starting dockercompose_ws_1
Starting dockercompose_cadvisor_1
Starting dockercompose_ws_2
Starting dockercompose_registrator_1
Starting dockercompose_haproxy_1
```

Figure 8: Container Initialization Result

To evaluate the results of container reactive scaling, the candidate assigns CPU load stress and records concurrent scaling out of web services. Subsequently removal of CPU load from web services, it is clearly evident of scaling in web services.

(i) For CPU load generation, the tradition of stress cpu 8 -v timeout 60 or yes/dev/null surges the number of hits reflected in CPU usage rate, consequently CPU load is increased. The results of scaling out of containers is shown in Figure 9. The below results refer con-

```
ubuntu@ip-172-31-7-248:~/dockerCompose$ sudo docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
b33ac571f17a        myws                   "/bin/sh -c 'php-f..."   1 second ago        Up Less than a second   80/tcp
                    dockercompose_ws_10
3b6db7d90e13        myws                   "/bin/sh -c 'php-f..."   16 seconds ago      Up 15 seconds           80/tcp
                    dockercompose_ws_9
ac44c608ece4        myws                   "/bin/sh -c 'php-f..."   31 seconds ago      Up 30 seconds           80/tcp
                    dockercompose_ws_8
ec12f890bdf0        myws                   "/bin/sh -c 'php-f..."   46 seconds ago      Up 45 seconds           80/tcp
                    dockercompose_ws_7
2859b0099265        myws                   "/bin/sh -c 'php-f..."   About a minute ago  Up About a minute       80/tcp
                    dockercompose_ws_6
a7046ade86ba        myws                   "/bin/sh -c 'php-f..."   About a minute ago  Up About a minute       80/tcp
                    dockercompose_ws_5
e9c4f0d42aa7        myws                   "/bin/sh -c 'php-f..."   About a minute ago  Up About a minute       80/tcp
                    dockercompose_ws_4
4601d9ae8fe3        myws                   "/bin/sh -c 'php-f..."   About a minute ago  Up About a minute       80/tcp
                    dockercompose_ws_3
99fa59266411        myproxy                "/bin/sh /start.sh"      2 days ago          Up 8 minutes            0.0.0.0:80-81->80-81/tcp
                    dockercompose_haproxy_1
def98415fad4        gliderlabs/registrator "/bin/registrator ..."   2 days ago          Up 8 minutes
                    dockercompose_registrator_1
4c006fd1ff82        myws                   "/bin/sh -c 'php-f..."   2 days ago          Up 8 minutes            80/tcp
                    dockercompose_ws_1
f6a4475887f6        monitoring             "/bin/sh /start.sh"      2 days ago          Up 8 minutes
                    dockercompose_monitoring_1
ccd02c4c0b2d        consul                 "docker-entrypoint..."   2 days ago          Up 8 minutes            8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.
0:8500->8500/tcp    dockercompose_consul_1
9ed912858e60        myws                   "/bin/sh -c 'php-f..."   2 days ago          Up 8 minutes            80/tcp
                    dockercompose_ws_2
ac52f6ed3e9e        google/cadvisor:latest "/usr/bin/cadvisor..."   2 days ago          Up 8 minutes            8080/tcp, 0.0.0.0:9091->9090/tcp
                    dockercompose_cadvisor_1
714bdeed325b        tutum/influxdb         "/run.sh"                2 days ago          Up 8 minutes            0.0.0.0:8083->8083/tcp, 8090/tcp, 0.0.0.0:8086->8086/tcp, 8099/tcp
                    dockercompose_influxsrv_1
```

Figure 9: Containers Reactive Scaling Out Result

tainer scaling out has been performed grounded on CPU utilization reactive threshold limit in which 10 containers scaled out from 2 containers. To validate the results, the candidate detected container CPU usage from cadvisor dashboard that exemplifies container CPU usage rate on top of individual CPU core rate. Built on consul information, HAproxy stats results are shown in Figure 10, it is evident that workload is balanced in accordance to round robin algorithm besides CPU usage of containers has stretched maximum level.

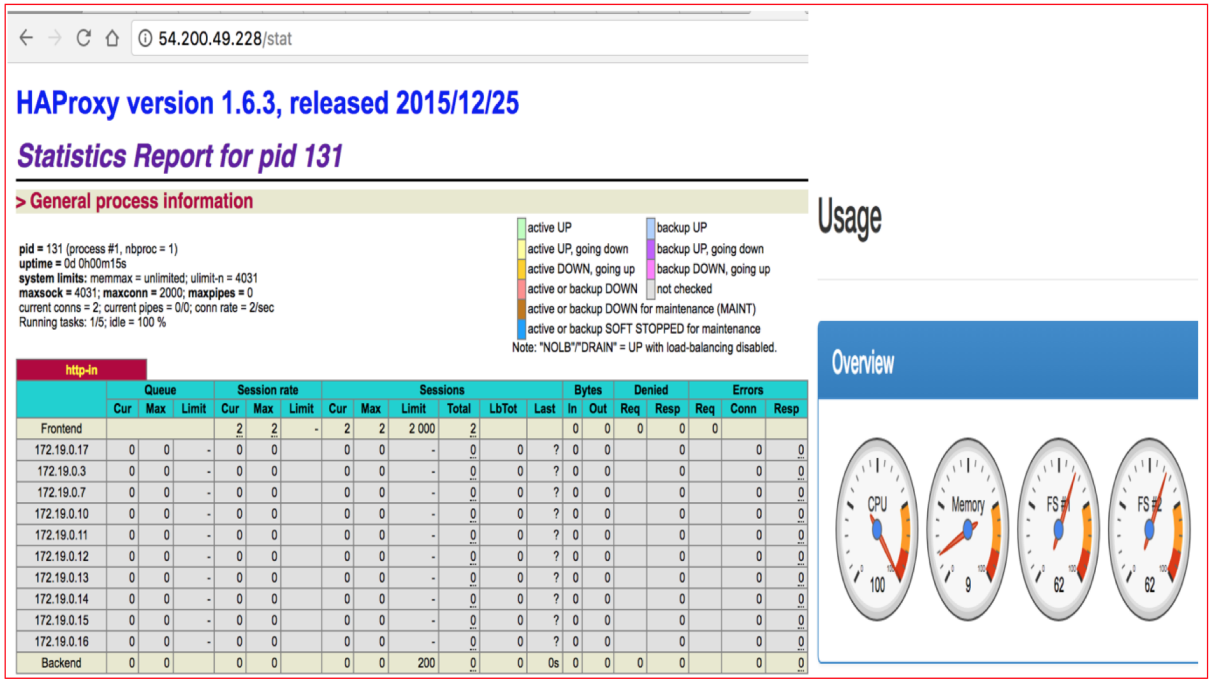


Figure 10: Containers Reactive Scaling Out Evaluation

(ii) To evaluate the reactive scale in of docker container web services, the candidate detached the CPU load using kill-all yes command. It abridged CPU usage of containers which is evident when CPU usage is reduced and the container web services gets decreased as depicted in Figure 11.

```
ubuntu@ip-172-31-7-248:~/dockerCompose$ sudo docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
99fa59266411        myproxy                "/bin/sh /start.sh"      2 days ago          Up 25 seconds       0.0.0.0:80-81->80-81/tcp
                    dockercompose_haproxy_1
def98415fad4        gliderlabs/registrator "/bin/registrator ..."   2 days ago          Up 25 seconds
                    dockercompose_registrator_1
4c006fd1ff82        myws                   "/bin/sh -c 'php-f..."   2 days ago          Up 26 seconds       80/tcp
                    dockercompose_ws_1
f6a4475887f6        monitoring             "/bin/sh /start.sh"      2 days ago          Up 26 seconds
                    dockercompose_monitoring_1
ccd02c4c0b2d        consul                 "docker-entrypoint..."   2 days ago          Up 26 seconds       8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:850
0->8500/tcp         dockercompose_consul_1
9ed912858e60        myws                   "/bin/sh -c 'php-f..."   2 days ago          Up 26 seconds       80/tcp
                    dockercompose_ws_2
ac52f6ed3e9e        google/cadvisor:latest "/usr/bin/cadvisor..."   2 days ago          Up 26 seconds       8080/tcp, 0.0.0.0:9091->9090/tcp
                    dockercompose_cadvisor_1
714bdeed325b        tutum/influxdb         "/run.sh"                2 days ago          Up 27 seconds       0.0.0.0:8083->8083/tcp, 8090/tcp, 0.0.0.0:8086->8086/tcp, 8099/tcp
                    dockercompose_influxsrv_1
```

Figure 11: Container Reactive Scaling In Result

The below HAproxy stats and CPU usage rate is perceived as soon as container reactive scaling in is achieved. It is palpable that containers scaled in while HAproxy distributed workload to 2 containers, in addition CPU usage is diminished to a minimum level and it is represented in Figure 12.

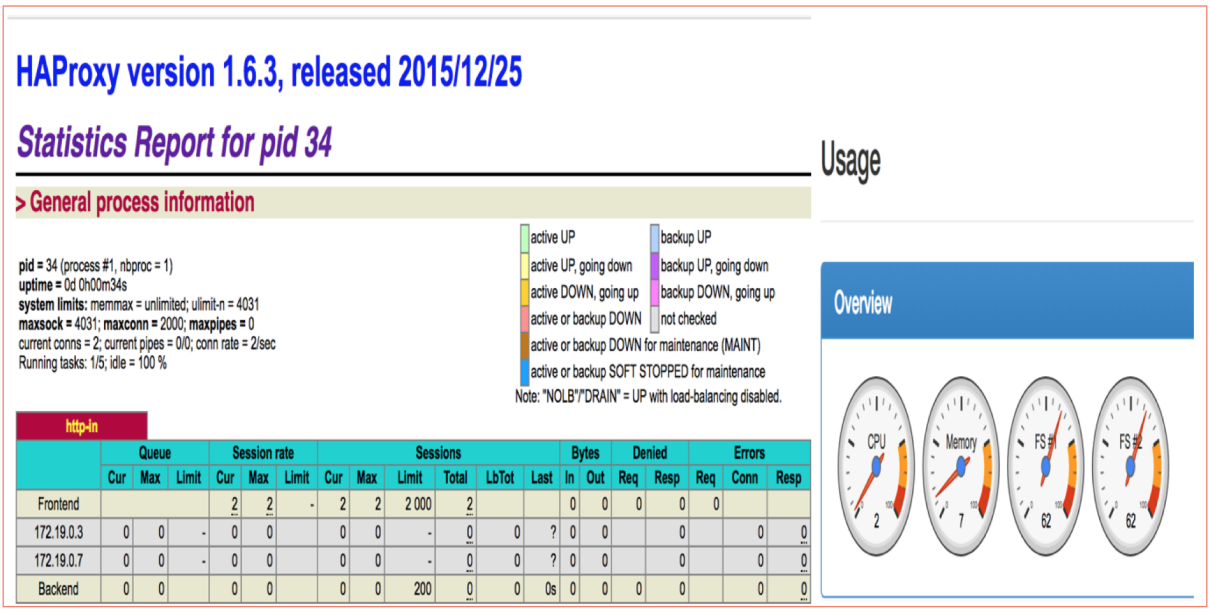


Figure 12: Containers Reactive Scaling In Evaluation

The overall CPU usage per core is visually shown in Figure 13. It is apparent that CPU usage is increased during load stress and gradually decreased on the end. The influxdb is associated with cadvisor and data metrics are internally stored in the database. The internal database metrics remains as not retrieved for graphical representation.

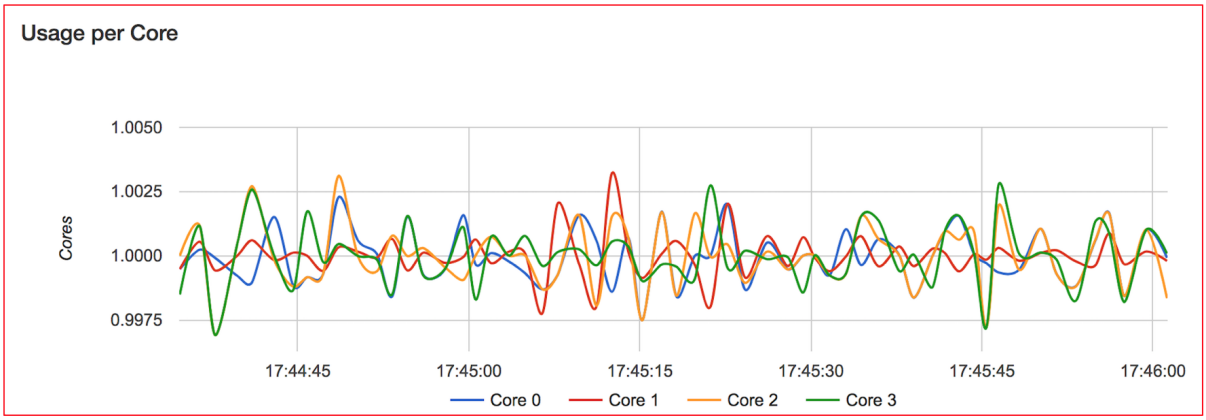


Figure 13: CPU Usage Per Core

Based on the results and evaluation on docker container reactive scalability, the research question specified in section 1.2.1 has been answered.

5.2.2 Evaluation and Results of Proactive ARIMA Model

(i) The CPU utilization data which is observed on AWS is used to perform prediction. [1] The visualization of CPU utilization of data prior to prediction is graphically shown in Figure 14. The graph illustrates the mean CPU usage that is calculated daily using resample function between 2014 May to 2014 July.

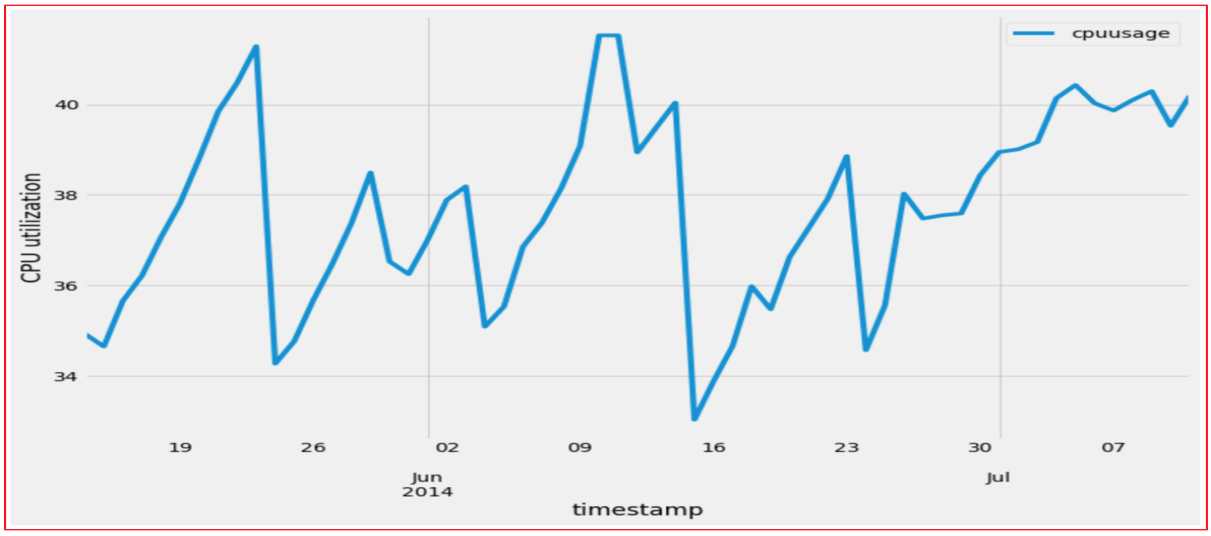


Figure 14: Data Visualization before prediction

(ii) The ARIMA parameter values is reserved made on the range (0, 2). Subsequently the data is dependent on time series, seasonal ARIMA time series model is used to test the seasonality. ARIMA values are tested and applied to SARIMA parameter. The models are trained using grid search to experiment several parameter combinations. Both ARIMA and SARIMA parameter combinations shaped the number of possible outputs. The candidate castoff AIC value using stats models to get good possible data output which is used to predict the future CPU utilization. In addition, list of combination results is tested and (1,1,1) parameter best data value 224.90 is second-hand in ARIMA model. The results are shown in Figure 15.

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.3146	0.156	2.011	0.044	0.008	0.621
ma.L1	-0.9335	0.144	-6.479	0.000	-1.216	-0.651
ar.S.L1	0.3146	0.156	2.011	0.044	0.008	0.621
ma.S.L1	-0.9335	0.144	-6.479	0.000	-1.216	-0.651
sigma2	2.7761	0.783	3.543	0.000	1.241	4.312

Figure 15: ARIMA Best Parameter Result

(iii) From results, the candidate performed diagnostic analysis to evaluate the results. The standard residual gives the deviation in overall CPU utilization. In addition, the normal distribution is assessed built on histogram and normal Q-Q plot test. The histogram bins are built on data results and estimated density of CPU usage is slightly differs

[1] https://www.kaggle.com/boltzmannbrain/nab

0.2 value from normality mean and standard deviation function. The standard residual and histogram graphs are showed in Figure 16.

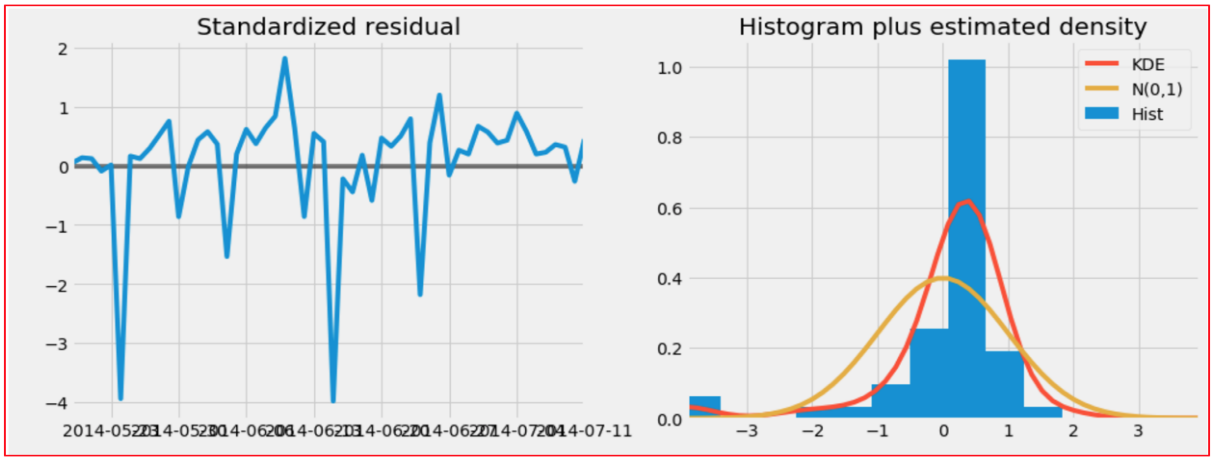


Figure 16: CPU Usage Residual and Histogram Evaluation

(iv) The normal Q-Q test and correlogram graphs are represented in Figure 17. From the graph, the residual distribution of CPU usage blue points is deviated from the linear CPU usage data. The correlogram graph illustrates the seasonality of time series since it illustrates less correlation value that is close to 0. The estimated results slightly diverged from normal distribution in the meantime data exhibits non-linearity and p value results remain less than z value i.e. 0.05.

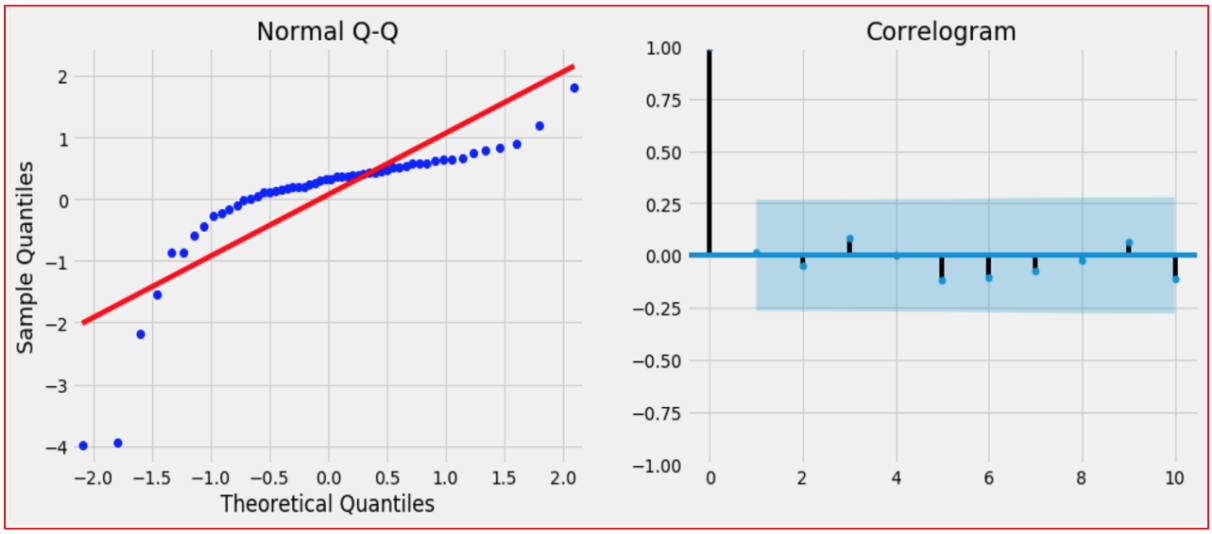
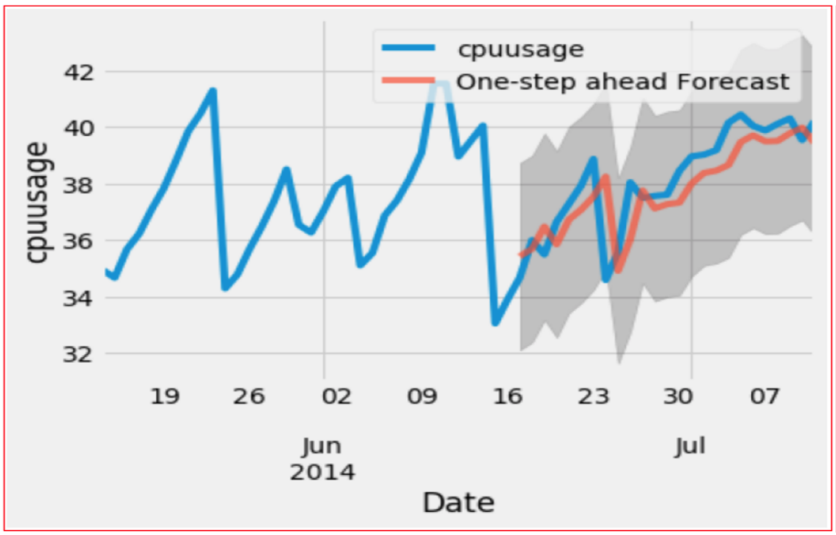


Figure 17: CPU Usage Normal Q-Q and Correlogram Evaluation

(v) One-step forward forecast of CPU usage is represented in Figure 18. The predicted results are detected till end of the July. It is obvious that predicted results are close to CPU usage. It predicts CPU usage value of 40 percent from the 7th July till the end.

(vi) The forecasting of CPU usage is depicted in Figure 19. The graph re-illustrates the forecasting trend for August months CPU usage shadowed by normal data. It is



timestamp	cpuusage
2014-06-17	34.663688
2014-06-18	35.987333
2014-06-19	35.477274
2014-06-20	36.632253
2014-06-21	37.270983
2014-06-22	37.918358
2014-06-23	38.858576
2014-06-24	34.573240
2014-06-25	35.562465
2014-06-26	38.030271
2014-06-27	37.479965
2014-06-28	37.551104
2014-06-29	37.591149
2014-06-30	38.436410
2014-07-01	38.949316
2014-07-02	39.008642
2014-07-03	39.168566
2014-07-04	40.137122
2014-07-05	40.427351
2014-07-06	40.026538
2014-07-07	39.865104
2014-07-08	40.097757
2014-07-09	40.292569
2014-07-10	39.528253
2014-07-11	40.188833

Figure 18: One-Step Further Forecast CPU Usage

apparent that forthcoming CPU usage displays the increase in trends and adjacent to average of 45 percent CPU utilization for upcoming months.

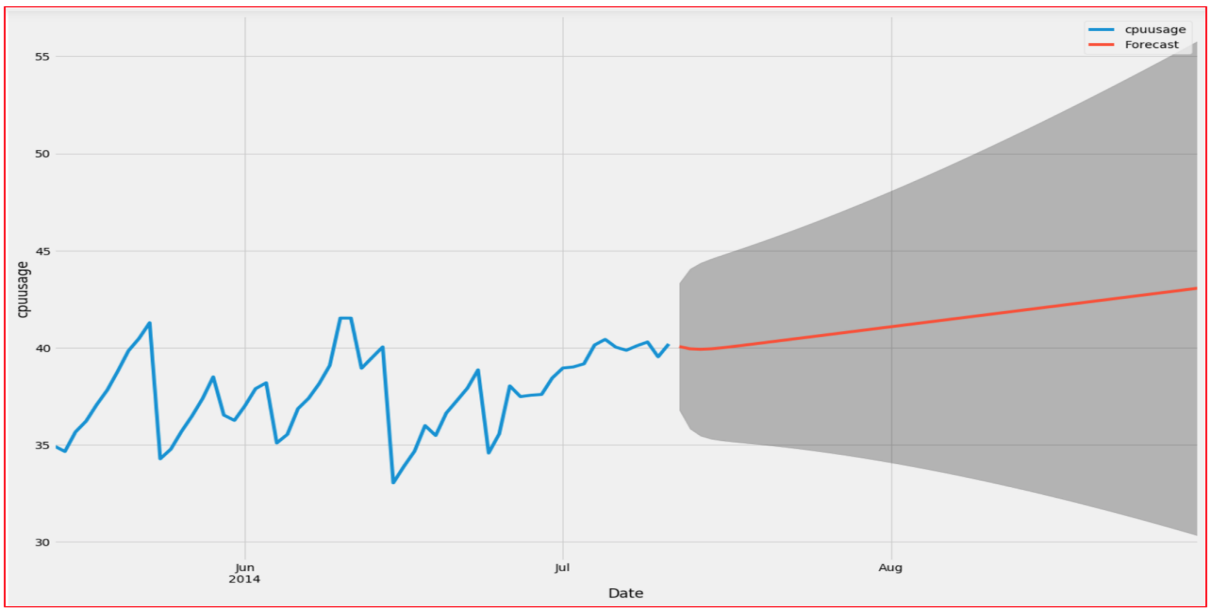


Figure 19: Forecasting CPU Usage

5.3 Conclusion

Based on the results and evaluation of CPU usage using proactive ARIMA model, the sub research question specified in section 1.2.1 has been answered and all stated objectives in section 1.2.2 has been tackled. The final results justify the purpose of this project that is to study and perform docker container reactive scalability and prediction forecast of CPU utilization by means of proactive models.

6 Conclusion and Future Work

Docker Containers are used extensively to run applications irrespective of platforms. Container scalability receipts the advantage to handle the applications seamlessly. The project work contributes docker container reactive scalability and prediction built on CPU utilization using proactive ARIMA model. The container metrics plays a pivotal role in scaling out or scaling in the required number of containers based on users demand. The approach is grounded on cloud reactive threshold-based policy and we adapted on containers. The containerized web service, load balancer and additional techniques are dockerized and achieved experiments on container reactive scalability based on CPU load stress. It generates real-time demand for container web service. The results of observation and evaluation has been done for horizontal reactive scaling on docker containers. Container reactive scaling HAproxy stats and cadvisor provided the scaling functions result based on CPU threshold. Furthermore, the proactive ARIMA time-series model is performed to predict and forecast the future CPU usage by using jupyter notebook . The CPU average usage on daily basis is visualized and parameter combination for ARIMA model results are illustrated. The ARIMA model is trained and evaluated through seasonal ARIMA by means of AIC values. The diagnostic analysis is evaluated and finally ARIMA prediction forecast of CPU usage is graphically depicted. The **future work** will be carrying out container reactive scaling on real-time application and analyse data prediction built on long term CPU utilization metrics can be obtained via back-end database. It can be achieved on integrating containerized ARIMA by real-time metrics database to record CPU usage gathered from large-scale applications.

Acknowledgements

I would like to thank the college staff for all the support during my MSc. I would also like to thank Dr. Catherine Mulwa (Supervisor) for her guidance throughout the project and my friends and family for their patience and support over the past one year.

References

Alzahrani, E. J., Tari, Z., Zeephongsekul, P., Lee, Y. C., Alsadie, D. and Zomaya, A. Y. (2016). Sla-aware resource scaling for energy efficiency, *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 852–859.

Balaji, M., Rao, G. S. V. and Kumar, C. A. (2014). A comparitive study of predictive models for cloud infrastructure management, *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 923–926.

Baresi, L., Guinea, S., Quattrocchi, G. and Tamburri, D. A. (2016). Microcloud: A container-based solution for efficient resource management in the cloud, *Smart Cloud (SmartCloud), IEEE International Conference on*, IEEE, pp. 218–223.

Calheiros, R. N., Masoumi, E., Ranjan, R. and Buyya, R. (2015). Workload prediction

using arima model and its impact on cloud applications x2019; qos, *IEEE Transactions on Cloud Computing* **3**(4): 449–458.

de Abranches, M. C. and Solis, P. (2016). An algorithm based on response time and traffic demands to scale containers on a cloud computing system, *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 343–350.

Dhib, E., Zangar, N., Tabbane, N. and Boussetta, K. (2016). Impact of seasonal arima workload prediction model on qoe for massively multiplayers online gaming, *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 737–741.

Erradi, A. and Kholidy, H. A. (2016). An efficient hybrid prediction approach for predicting cloud consumer resource needs, *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–8.

Hegde, A., Ghosh, R., Mukherjee, T. and Sharma, V. (2016). Scope: A decision system for large scale container provisioning management, *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 220–227.

Hu, Y., Deng, B. and Peng, F. (2016). Autoscaling prediction models for cloud resource provisioning, *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1364–1369.

Inagaki, T., Ueda, Y. and Ohara, M. (2016). Container management as emerging workload for operating systems, *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–10.

Kan, C. (2016). Docloud: An elastic cloud platform for web applications based on docker, *2016 18th International Conference on Advanced Communication Technology (ICACT)*, pp. 478–483.

Kim, W.-Y., Lee, J.-S. and Huh, E.-N. (2017). Study on proactive auto scaling for instance through the prediction of network traffic on the container environment, *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, IMCOM '17, ACM, New York, NY, USA, pp. 17:1–17:8.

Meng, Y., Rao, R., Zhang, X. and Hong, P. (2016). Crupa: A container resource utilization prediction algorithm for auto-scaling based on time series analysis, *2016 International Conference on Progress in Informatics and Computing (PIC)*, pp. 468–472.

Moradi, F., Flinta, C., Johnsson, A. and Meirosu, C. (2017). Conmon: An automated container based network performance monitoring system, *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 54–62.

N, P. E., Mulerickal, F. J. P., Paul, B. and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization, *2015 International Conference on Control Communication Computing India (ICCC)*, pp. 697–700.

Zhang, H., Ma, H., Fu, G., Yang, X., Jiang, Z. and Gao, Y. (2016). Container based video surveillance cloud service with fine-grained resource provisioning, *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 758–765.