IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

The Performance Measurement of Containerized Applications based on Micro Services Architecture

Dr. Kamlendu Kumar Pandey
Veer Narmad South Gujarat University
Surat, Gujarat

Abstract

Micro Services Architecture I.e "MSA" are the the new paradigm of software engineering and development. This is complete deviation from current client-server and monolithic architecture and emphasises on small and smart containerised business resources which can e effortlessly deployed on cloud. Many such small resources running on cloud or group of clouds comminicate with each other to create an entire organisational business application. This paper is effort to measure the performance of such applications in a scalable and orchestrated environment. All current industry trends and tools are being followed in the application development. The measurement of performance is done in a controlled simulated environment. The strategy of performance measurement is worked out theoretically and is mesured practically in a controlled environment.

Keywords : Micro Services, containers, cloud, performance testing

1. Introduction

The current client server applications or applications deployed on monolithic architectures are facing acute problems of service latency, sacleability, manageability and requirment change hells. The applications designed with best programming languages and platforms fail to perform as per the requirement standards. The major issues are related to change in technology, deprecated apis, under perfornce of platforms, change of experienced development team with the new one. As almost all applications now a days are web or enterprise applications, they starting attractiving load in exponential proportions as the popularity of application grows. The current applications require more power to scale and perform which involves a very complex resource procurement and management strategies along with techbological complexity. A failure in achieving the task may result in downgraded performance, customer dissatisfaction and ultimately the business loss.

To deal with all these issues the technology of Service Oriented Architecture was introduced which used SOAP based webservices. SOA abstracts the complexities implementation of language based business logic and protocol based communications and gives a very easy to use plug and play end points to work within the application. Using the loos coupling and a Decomposed Business model it solves many teething problems of current enterprise applications. The best thing is that that this architecture is well standardized by W3C and is a part of reference implementation in almost all major players of software industry. The standardization solves the compatibility issues with in the vendors and standards so developers have to focus mainly on the the orchestration part rather than peeking into the code and fine tuning it as per new standards. The orchestration tools and the Business Process Execution Language for Web Services (BPEL4WS) makes it most attractive option to work with. It facilitate point to point, multi point communication with sufficient flags and fault handlers creating a composite application involving services abstracting the business logic written in various languages. Almost all the vendors are providing the XML interface or GUI based wizards . Not only business logic but even system based events like signals, messaging, authentication and authorization repositories can also be abstracted as a service. Although SOA was promising and many efforts were done in this direction to implement SOA but it could not give the results as expected. There were several resaons for thar. First the developers have to learn some new languages like BPEL4WS for service linking and compositions, secondly most of the implementation as proprietry. The addition of Enterprise Service Bus was so difficult and complex to configure that most of the software companies preferd to stick with the old trends as safe bet and non distruptive. ESB was implemented as additional layer on the existing web servers and that made the webservers bulky and comples to configure and to work with. You need to build an entire SOA organisation to implement it effectively.

Microservices came as a complete alternative solution to tackle all the above mentioed issues. Rather tah building huge applications, the emphasis is now on bulding small or micro applications[1] for a specific need in the business organisation. The philosophy is to decompose the entire business domain tree into small sub domains and leflets (services). Instead of entire team working on a big project, small teams are assigned to these micro services. This team is responsible for its behavious and correctness in the service. A complete connectivity or a pipeline architecture is evolved where all these micro service will communicate with each other effectively using the light weight protocols like http and all the resources being accesed by HTTP methods[1][12] . These resources are called REST resourcces which are the end points to communicate with users or other services. The REST end points collect all the information and results from the models, entities and resources behind the scene. Apart from the structure and communications microservices architecture comes with additional features like Tracing, Health Checkup, Fault Tolerance, Scheduling, Data grids etc [2]. The development to production issues and gaps are solved by containerization technologies like Docker[7]. The scaling and availability issues are solved by new independent container orchestration technologies like **Swarm and Kubernetes.** The software development practice is also evolved with current trends as service repositories like GIT and piplining with application like Jenkins which are finally giving a complete Continuous Integration / Continous Delivery (CI/CD) based design pattern[11].Micro services come up as self manageable light weight component[6][8].

The emphasis of the paper is to come out with a model to test such applications for their performance. The factors which influence the performance are execution environment, intercommunication overhead, encryption and decryption of tokens and content, container scheduling, orchestration bottlenecks, instance preservation, failure and autocreation and accesibility. The paper is arranged as follows: section 2 deals with the functioning of the Micro Services Architecture, Section 3 depicts a real world Finance Companying problem which can be tested on Micro Services, Section 4 is about the assessment of the problem and working out an analytical solution of the load / stress on the system and thereby its performance under various constraints, Section 5 is real world stress testing of the application using popular web testing tools and thereby validating the developed model, Section 6 is about discussion and future work while Section 7 is the conclusion of the paper.

2. ABOUT MICRO SERVICES ARCHITECTURE

As the paper is dealing with performance testing model of Micro Services Architecure it is imperative to know its structure , components, functioning , interaction, events and transactional aspects so that all the factors are wisely considers in the underlying example application. Letus what the transformation from a traditional Monolithic Application to Micro service application look like.

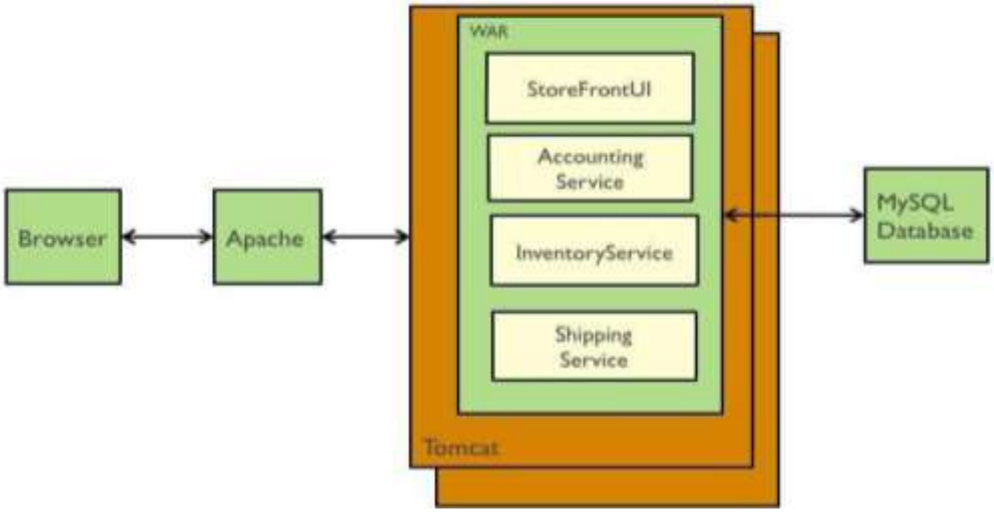


Fig.1 A monolithic Application (courtesy www.microservice.io[14])

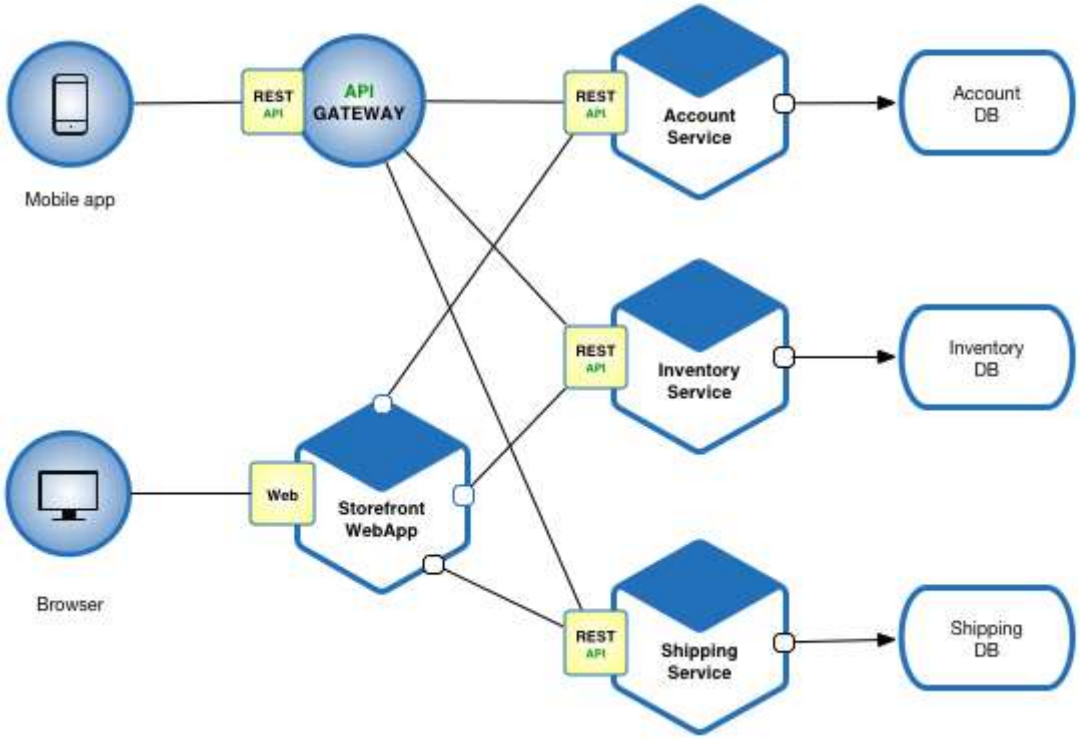


Fig.2 Monolith decomposed into micro services (courtesy www.microservices.io [3][14])

Looking to the fig.1 and 2 one can clearly make out the difference between a monolith and micro services architecture. In monolithic all the applications and layers are deployed into one application container. The major risk is if server containers fail or underform the entire application will go down which is not affordable. In fig.2 every service which is part of bigger application is running in its owner container system . If one service fails it is no way going to affect other services which are not connected with it. The next question comes that if a service fails then the application which is using this service will also fails. The solution is given by an active load balancer and a container node orchestrator which will create the container instance on its own if it fails. The Microservices architecture comes with following components.

1. Application
   1. UI/UX (User Interface (optional)
   2. Business Logic
   3. REST Resouce and End Points
   4. SSL based Authentication and Authorization with JWT (Json Web Tokens)
   5. Config
   6. Fault Tolerance
   7. Health
2. Micro Servers where applications can be deployed
3. Containers
4. Container Orchestrators
5. REST Orchestrators and Registry
6. API Gateways

Using all above we can develop complete spplication in microservices architecture. The steps to build are as under.

1. Decomposition of Domain into subdomains and utimately to leaflets called services
2. Code for the Application with entities, model , services and REST End points.
3. Build using a smart builder like MAVEN . The outputs are not conventional war files but jar, uber jars or hollow jar files which are extremely light weight.
4. Choose a server with minimal foot print and deploy your application with the server into a container.
5. Arrange all the services to execute in the predefined pattern using a pipelining application like Jenkins / Consul
6. Containerize the application using applications like Docker
7. Deploy containers into a container orchestrator like Kubernetes/ Swarm
8. Chooss an API gateway like Ingress / Kong and give urls
9. Deploy entire arrangement on clouds like Amazon / Google / Azure

to the client to browse the application and iteract with it.

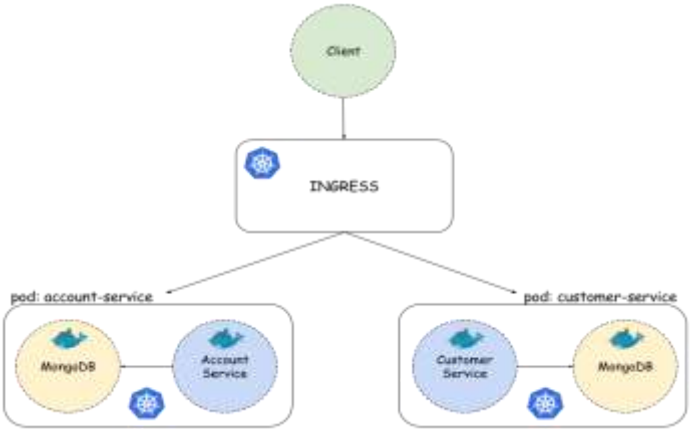


Fig .3 A schematic model of Micro Services

1. *Application :* The code must be written in a language and platorm which supports micro services like Jakarat EE 's Micro Profile **or Spring boot.**
2. *Package Builders :* Thin builders like **Maven** create an optimized build / packaging of the applications resulting in uber jars or hollow jars
3. *Pipe Liners :* Pipeliners like **Jenkins** arrange builds in a predefined manner to greate the whole application. The pipeliners are important part of Continuous Integration and Continuous Delivery (CI/CD) way automated software engineering.
4. *Containers* : The containers like **Docker** create complete virtualized environment with server and application ready for production and can run on any Operating System without having to install a virtual machine. This decreases a lot of load on the Underlying OS or bare metal server. The containers can talk to each other with underlying bridge networks or overlay networks.
5. *Container Orchestrators* : , they are required for manay management tasks like load balancing, dynamic instance creation or destruction based on load . **Kubernetes** is one such orchestrator.
6. *API Gateways*: API gateways are used to redirect the request of customers based on the content of URL. API gateways like **Ingress** from Nginx are popular.
7. *Cloud* : All these services are also created by popular clouds like **AWS or Azure** and one who subscribes the cloud can all the above things in built in the for example Amazon has got services like Elastic Compute, Elastic Container System (like Docker repositories), Elastic Kubernetes Services (EKS) which allow us to create a kubernetes cluster. Amazon Load Balancer (ALB) is the API gateway. S3 services are to store all your data and files.
8. *Performance Monitor* : Specia tools like **Prometheous** are used to monitor the health of the application and constantly monitor the performance of communication channels. They have a dashboard system to show the graphical charts showing the performance. A comprehensive reporting tool has been developed to take the trace and event incase of failures.

3. THE APPLICATION FOR PERFORMANCE EVALUATION

Scenario : For testing the performance of business application based on micro services platform , a Finance Company mortgaging application is developed where a Finance Company classifies the customer' s eligibility for the application of Mortgage Loan. The Finance Company provides a client interface for applying Mortgage Loan and the client applies by providing the data asked by the the Finance Company . The Finance Company later on classifies the applicants. There are two criteria of classification 1) on the basis of desired interest rate and secondly on the basis of the amount. The Finance Company have three sanctioning authorities for Mortgage Loans based on the gradation of the customers. If the customer applies for a Mortgage Loan of an amount under a certain threshold limit and as per the interest rate charged by the Finance Company, such customers are subjected to a Normal Mortgage Loan processing service where decision can be taken by the Mortgage Loan officer. The customers who have demanded a lower charge of interest rate are subjected to the approval of Manager who depends upon the past experience has the right to approve or disapprove. If the customer applies for a larger Mortgage Loan above a threshold limit then the approval process is complex. In that case a their party financial service is utilized to obtain the credit rating of the customer. If the credit rating of the customer is above certain grade then the Mortgage Loan for that customer can be approved else rejected.

ALGORITHM :

Process :

```
1. Finance Company Threshold values :
    PIR : prevailing interest rate of Finance Company
    TLA: Threshold Mortgage Loan amount
    MCR : Minimum credit rating on Large Mortgage Loan amount
    CCR : Customer Credit Rating
    VCA : Valid Customer Application (True, False)
2. Mortgage Loan Client applies for  Mortgage Loan
    var MortgageLoanRequest  : properties : cname, caddress, cutomerMortgageLoanAmount, AskedIntrestRate
3. Finance Company Receives the Applications
    call ApplicationCheckLogic
    if VCA is True  then
        if customerMortgageLoanAmount LESS THAN  TLA
            If AskedIntrestRate LESS THAN PIR
                Call NormalMortgageLoanProcessing Service
            Else
                Call ManagerMortgageLoanProcessing Service
        Else
            Call LargeMortgageLoanProcessing Service and
            FCR = call populateFCR
            If  FCR More than CCR Then
                Goto acceptMortgageLoan
            Else Go to rejectMortgageLoan
    End If
```

Entities :

Customers
MortgageLoanMaster
MortgageLoanApplication

Business Logic :

: ProcessNormalMortgageLoan
: ProcessManagerMortgageLoan
: ProcessLargeMortgageLoan
: populateFCR
: ValidateApplicationLogic

Messages :

:acceptMortgageLoan : “Your MortgageLoan is approved:
:rejectMortgageLoan : “Your MortgageLoan is rejected”

The above scenario and the algorithm clarifies the problem. This scenario gives us following services to constructed to abstract the business logic

Rest Resources :

1. NormalMortgageLoanResource	: End Point of logic of NormalMortgageLoanProcessing
2. ManagerMortgageLoanResource	: End Point of the logic of ManagerMortgageLoanProcessing
3. LargeMortgageLoanResource	: End Point of the logic of LargeMortgageLoanProcessing
4. CreditRatingResource	: End Point of the logic of populateCCR (Third party)
5. MortgageLoanvalidationResource	: End Point of the logic of ValidateApplicationLoic

Client handles

1. ValidationResource	: End Point of MortgageLoanApp in the algorithm for validation
2. MortgageLoanAppResource	: End Point of leading to MangerMortgageLoanService
3. LargeMortgageLoanReqResource	: End Point of leading to Normal or large MortgageLoan processing

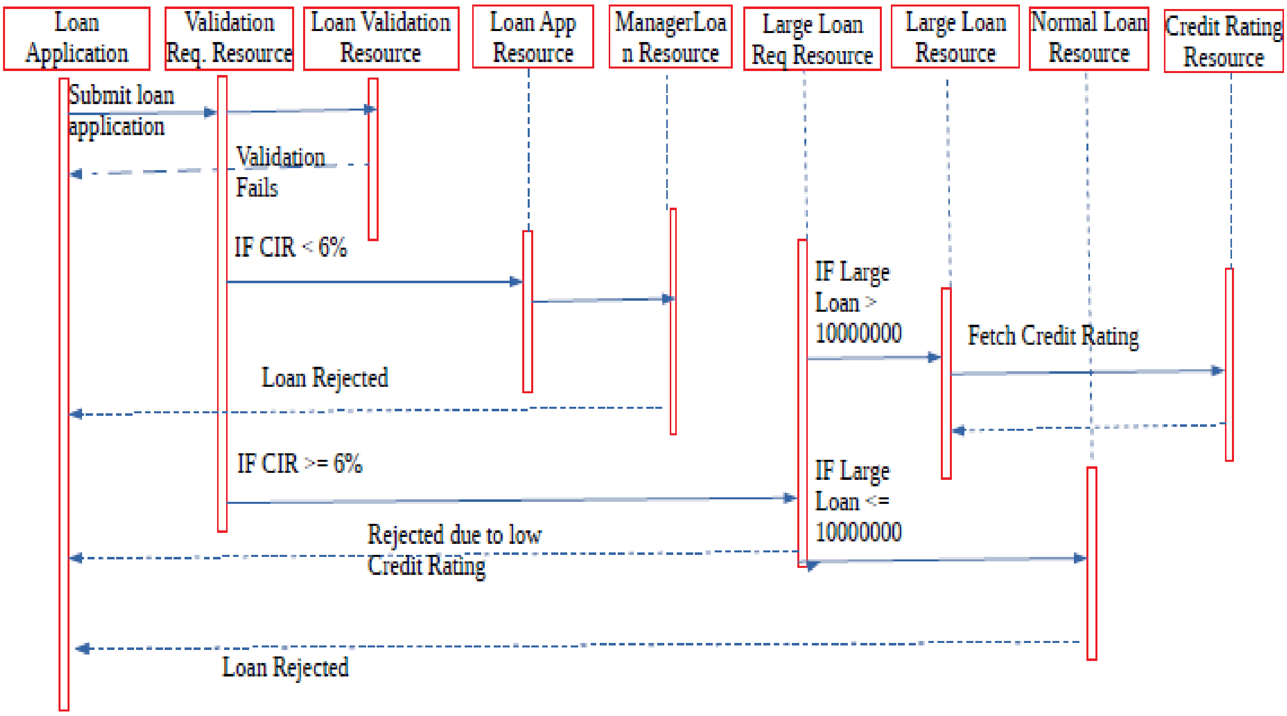


Fig. 4 : Sequence diagram of the application.

The Services are correctly identified in this model . The application was developed in Oracle Fusion Middleware and Oracle Service Bus which developed over Weblogic Server. The IDE used for developing the microservices is NetBeans. The console is a web based application which facillitates the developers to do all the functionalities like transformation, routing, pipe lining, altering the message structures and service call out. The application was tested for its functioning with test data. The input and output formats were JSON artifacts.

4. The Testing Strategy

Now as we have already identified the crucial service and steps to test the applications we need to find out the factors responsible for the performance of the ESB. Two types of systems exist. The close system and the open system. The close system highly synchronous and a predicatble corelation can be esatblished by the participating logic. Various testing tools are devised for that. The Open System has a quasi way of execution as many request are arriving in concurrent and asysnchronous matter[15]. The our application is a open system and thus is very difficult by predicting it through the analytics of the closed model[15]. Several researchers have tried to use closed model for this application and compared the result with the real testing tool. There is still no reliable analytical solutions for open system. Lets identify the time parameters for various services and their branches of execution.

Time Parameters

Business Resources :

1. NormalMortgageLoanResource : TNLR
2. ManagerMortgageLoanResource : TMLR
3. LargeMortgageLoanResource : TLLR
4. CreditRatingResource : TCR
5. MortgageLoanvalidationResource : TVAR

Client Resources

1. ValidationProxyResource : TVPR
2. MortgageLoanAppProxyResource : TNPR
3. LargeMortgageLoanProxyResource : TLP

Any additional time in sending request and receiving response : Ta

Table 1 : Branch Time Equations

No.	Process	Time Estimate
1	MortgageLoan Application with validation failure	TVPR = TVAR + Ta
2	MortgageLoan Application with validation success and IR> 6	TVPR = TVAR+ TMLR+Ta
3	MortgageLoan Application with validation success and IR> 6 and Amount less than TA	TVPR = TVAR+ TNR+Ta
4	MortgageLoan Application with validation success and IR> 6 and Amount less than TA	TVPR = TVAR+ TLLR+ TCR +Ta

In all the above equations the uncertainty is in the value of Ta which may vary as per the location of requester. The test results can be put into the equation and iteratively we can work out the load account on various services under various conditions of the request. This will give a reliable estimate for preparing a reasonable analytical solution for an open ended system.

The test results can be compared with the estimated demand from the literature

Table 2. Estimated demand for the services from literature (for the closed system) chapter 9 [6]

Service name	Service type	Load (ms)
TVPR	Client	2.41
TLNR	Client	3.28
TLPR	Client	5.91
TMLR	Resource	97.28
TVLR	Resource	68.44
TNLR	Resource	110.35
TLLR	Resource	156.80

The test was done on the application with varied parameters and no of concurrent request by increasing the number of clients. The throughput of the test are recovered from the logs and compared with the analytical throughput.

5.0 RESULTS AND DISCUSSION

5.1 The Test Experimental Setup

To conduct the testing of various services and branch operation the application was developed on the following platform

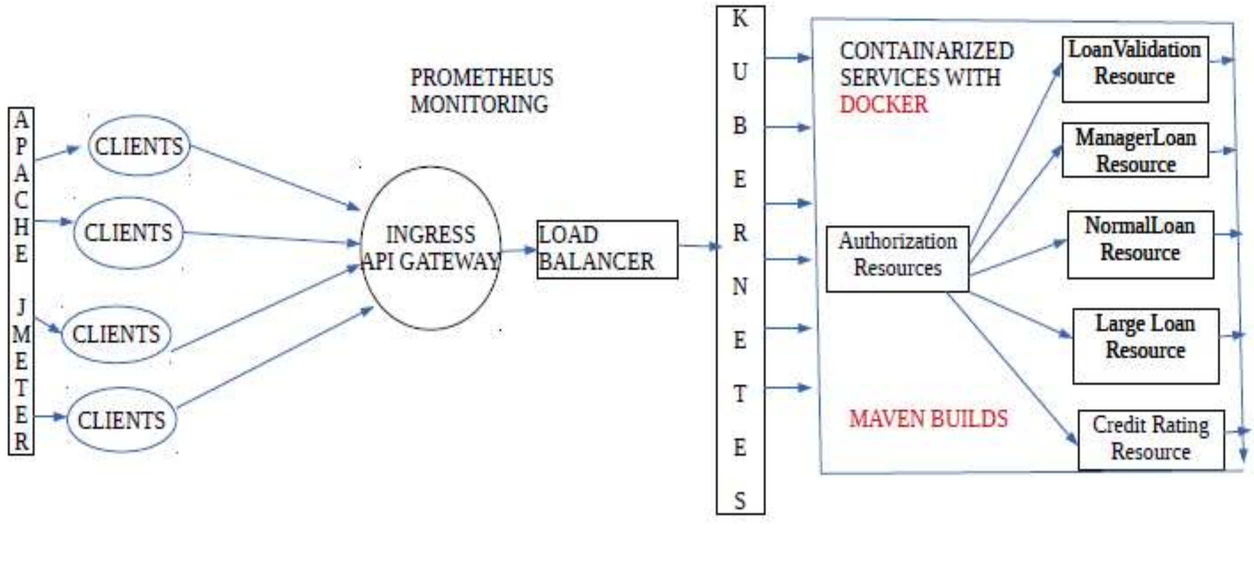


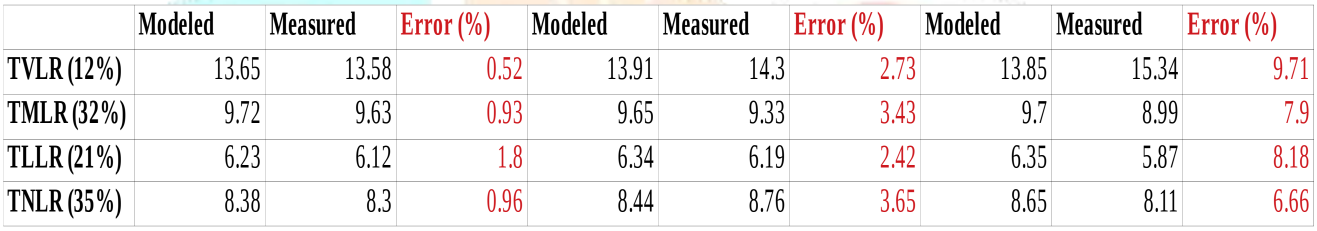
Fig. 5 : Setup of the Experiment for test in consideration.

Table 3 : Test Setup

Hardware	Dell Server with 8 GB RAM quad core CPU 1.2 GHz, 1 Work station client on LAN
Software Development	Minikube Kubernetes Cluster, Docker, Ingress API gateway, Prometheus, Payara Micro Java EE Server
Programming language	Java Enterprise Edition 8
Testing Tool	*Apache Jmeter – A open source tool* Apache JMeter ise used to judge performance both kind of resources (dynamic and static), Web dynamic applications. It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load

The test was done on the application with varied parameters and no of concurrent request by increasing the number of clients. The throughput of the test are recovered from the logs and compared with the analytical throughput.

Table 3. Error of performance modeling



	Modeled	Measured	Error (%)	Modeled	Measured	Error (%)	Modeled	Measured	Error (%)
TVLR (12%)	13.65	13.58	0.52	13.91	14.3	2.73	13.85	15.34	9.71
TMLR (32%)	9.72	9.63	0.93	9.65	9.33	3.43	9.7	8.99	7.9
TLLR (21%)	6.23	6.12	1.8	6.34	6.19	2.42	6.35	5.87	8.18
TNLR (35%)	8.38	8.3	0.96	8.44	8.76	3.65	8.65	8.11	6.66

Above are the results obtained by running the test over various branches which seems to be quite in agreement with the analytical modeled results. The error is ranging between 1 to 10 % . Interesting results are obtained from the error matrix. We can see the error is under control when no of users are 50 but increasing the load somewhat deteriorates the performance and error goes up to 10% in case of Validation Service. The Normal Mortgage Loan Service are quite in control between the range of 4 to 5% while manage Mortgage Loan Service is ranging between 3 to 6%. These results can be used for creating a solution for open end system. We see that the time is not predictably in the proxy services as they are not executed in system environment but depends on lot many external factor. Substituting the time consumed by various branch outs and no of clients the analytical equations can be iteratively solved to obtain the right solution for open ended systems.

Several other factors are not considered here and can be done in the future work, The factors are Authorization and Authentication load . This is typical because now a days we have several ways of security mechanism from login-password to json web tokens or OAuth and OpenId connect. Such aspects are not yet covered in the analytical models. The other factor is the network firewall and devices like routers which carry your request to the target service or proxy services.

6. CONCLUSION

Container based Micro Service Architecture is current trend of development and a part of Service Oriented Architecturein the inductry. A model was developed for the micro services based on Representational State Transfer (REST), containerization done in docker plateform and its load balancing and orchestration done by Kubernetes. This application was developed for the purpose of testing using Jakara EE and payara micro. This application was tested for the stress or load in the LAN based architecture and a comparison was done to understand the analytical load calculation and the test results. The test validates the analytical model but in case of client calls the error increases as the no of current user increases. This paper is an effort as how to validate a open systems like Micro Services Using this one can work on developing a reliable model for REST based and containerized applications in a CI/CD model of software development.

References

[1] Hassan et al. 2017 Hassan, S., Ali, N., and Bahsoon, R. (2017). Microservice ambients:An architectural meta-modelling approach for microservice granularity. In IEEE In-ternational Conference on Software Architecture (ICSA), pages 1–10

[2] Alshuqayran et al. 2016 Alshuqayran, N., Ali, N., and Evans, R. (2016). A SystematicMapping Study in Microservice Architecture. In 9th International Conference onService-Oriented Computing and Applications (SOCA), pages 44–51.Fowler and Lewis 2017 Fowler, M. and Lewis, J. (March 25, 2014.

[3] Klock et al. 2017 Klock, S., Werf, J. M. E. M. V. D., Guelen, J. P., and Jansen, S. (2017).Workload-based clustering of coherent feature sets in microservice architectures. InIEEE International Conference on Software Architecture (ICSA), pages 11–20

[4] Mori et al. 2018 Mori, A., Vale, G., Viggiato, M., Oliveira, J., Figueiredo, E., Cirilo, E.,Jamshidi, P., and Kastner, C. (2018). Evaluating domain-specific metric thresholds: anempirical study. In International Conference on Technical Debt (TechDebt).Newman 2015 Newman, S. (2015). Building Microservices. O'Reilly Media, Inc.

[5] Oliveira et al. 2018 Oliveira, J., Viggiato, M., Santos, M., Figueiredo, E., and Marques-Neto, H. (2018). An empirical study on the impact of android code smells on resourceusage. In International Conference on Software Engineering & Knowledge Engineer-ing (SEKE).Pahl 2015 Pahl, C. (2015). Containerization and the PaaS Cloud. IEEE Cloud Computing,2(3):24–31.

[6] Aderaldo et al. 2017 Aderaldo, C. M., Mendonc¸a, N. C., Pahl, C., and Jamshidi, P. (2017).Benchmark Requirements for Microservices Architecture Research. In 1st Interna-tional Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), pages 8–13.

[7] Pahl and Jamshidi 2016 Pahl, C. and Jamshidi, P. (2016). Microservices: A systematicmapping study. In 6th International Conference on Cloud Computing and ServicesScience (CLOSER), pages 137–146.

[8] Ramos et al. 2016 Ramos, M., Valente, M. T., Terra, R., and Santos, G. (2016). Angu-larJS in the wild: A survey with 460 developers. In 7th International Workshop onEvaluation and Usability of Programming Languages and Tools, pages 9–16.

[9] Markos Viggiato1, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, Eduardo Figueiredo Microservices in Practice: A Survey Study

[10] Online; accessedApril, 2017). Microservices. https://martinfowler.com/articles/microservices.html.Guimaraes et al. 2013 Guimaraes, E., Garcia, A., Figueiredo, E., and Cai, Y. (2013). Pri-oritizing software anomalies with software etrics and architecture blueprints: a con-trolled experiment. In Proceedings of the 5th International Workshop on Modeling nSoftware Engineering, pages 82–88. IEEE Press.

[11] S. Newman. Building Microservices. O'Reilly Media, Inc., 2015.

[12] M. Fowler and J. Lewis. Microservices a definition of this new architectural term.URL:http://martinfowler.com/articles/microservices.html

[13] Chris Richardson and Floyd Smith , Microservices : From Design to development

[14] [online] http://microservices.io maintained by Chris Richardson

[15] -Bianca Schroeder, Adam Wierman and Mor Harchol- Balter. Open vs closed: a cautionary tale, Networked System Design and Implementation NSDI,2006.