A COST-EFFECTIVE AUTO-SCALING STRATEGY BETWEEN SERVER AND SERVERLESS ARCHITECTURES FOR CLOUD DEPLOYMENTS

LDSB Weerasinghe

(189357U)

Degree of Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2020

A COST-EFFECTIVE AUTO-SCALING STRATEGY BETWEEN SERVER AND SERVERLESS ARCHITECTURES FOR CLOUD DEPLOYMENTS

LDSB Weerasinghe

(189357U)

Thesis submitted in partial fulfillment of the requirements for the degree
Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2020

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.
Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ……………………. Date:……………….
Name: Sidath Weerasinghe

The supervisor should certify the dissertation with the following declaration.

The above candidate has carried out research for the Masters Dissertation under my supervision.

Signature of the supervisor: …………………………. Date: ………………..
Name: Dr. Indika Perera

ABSTRACT

Nowadays, cloud computing is marking as an emerging technology in the universe. Most of people use cloud environments to deploy their systems. Many enterprise systems give their benchmarks according to the specific instances in the cloud. Cloud providers provide primary services as SaaS, PaaS, IaaS and FaaS, then Cloud consumers use those services as per their demands.

The systems which are developed on several years ago are still running upon on-premise environments. They are tightly coupled legacy systems. Hence people cannot adopt new technologies to those monolithic systems. Nevertheless, people who are in the software industry need to integrate new technologies into their systems to stay competitive in the IT industry. To get into the new technology that should be cost-effective, more reliable, and should need to cater the new and existing functionalities of the system. One of the problems is that migrating the whole system or partial system to the new technology is hazardous. Sometimes the system needs to handle substantial traffic loads, according to the business needs. In order to satisfy sudden traffic spikes, systems need to scale horizontally or vertically. If their application server is deployed upon the on-premise server, scaling is complicated, and if the application is deployed on the cloud, VMs scaling is possible, but it can be very costly.

In this research, the goal is to overcome these scaling problems. The research component presents the strategy to load balancing the traffic between the server and serverless function. The cost of the serverless function calculated only for the executes requests, so it is cheaper than running a new server. Therefore this hybrid strategy is a highly cost-effective way to scaling the servers and also proven that the proposed system is capable of load balancing the traffic according to the server loads with the analytical results. Other researches are proven that this research component is the right solution for ongoing industry problems.

Keywords: Cloud Computing, Serverless, Load Balancing

ACKNOWLEDGMENTS

My sincere appreciation goes to my family for the continuous support and motivation given to manage my MSc research work. I also express my heartfelt gratitude to Dr. Indika Perera, my supervisor, for the supervision and advice given throughout to this research period. I also thank my batch mates, for their untiring help to find research materials. I am also thankful to WSO2 Telco (Pvt) Ltd, for providing support to complete this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SaaS	Software as a Service
PaaS	Platform as a Service
FaaS	Function as a Service
IaaS	Infrastructure as a Service
NRPE	Nagios Remote Plugin Executor
AWS	Amazon Web Services
GCP	Google Cloud Platform
TPS	Transitions Per Second

CHAPTER 1
INTRODUCTION

1.1 Preamble

Cloud computing is a rapidly emerging technology in today's era. It is referred to as accessing computing service over the internet. Cloud computing came into its presence because of the evolution and endorsement of already existing paradigms and technologies. A recent quantitative survey by on the taking on rates of cloud computing technology by companies reported that 77% of large companies are using the cloud, although 73% of small and medium-sized companies are adopting the cloud[1].

The main object of using the cloud is to gain benefits from different Computing paradigms. The cloud objectives to reduce the costs and helps the clients to focus on their business rather than to care about the obstacles and barriers within the information technologies. There are have three main pillars in cloud computing. They are infrastructure as a service (IaaS), platform as a service (PaaS) and software as service (SaaS).

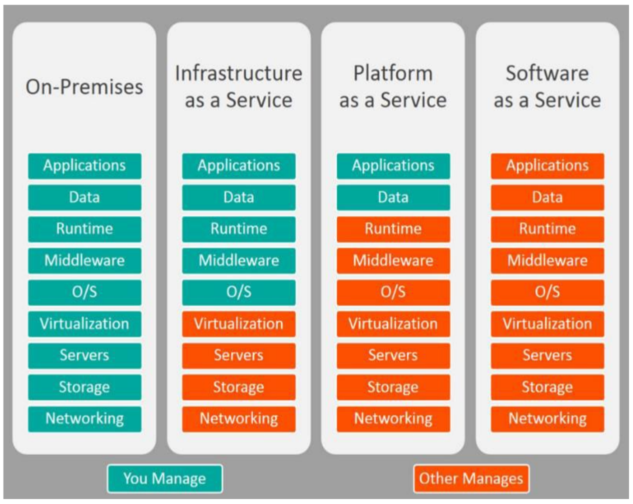


Figure 1 : Cloud computing stack

1.2 Motivation

This research shows that cost-effective way to handle the sudden heavy traffic between server and serverless. Most of the businesses run their applications upon on-premise servers or cloud servers. Nevertheless, most of those applications are tightly coupled monolithic applications. With the current practice, those monolithic applications need to horizontally scale for catering to the traffic spikes. That means every time there is a spike in traffic, those companies need human resources, server resources, and additional effects to satisfy the end-user demands. Considering the cost, the existing method is very costly, and it will take more time, such as doing a new production deployment, testing and so on.

1.2.1 Infrastructure as a Service (IaaS)

Most of the industries use this service to host their servers and maintain storage and several other infrastructures. Cloud consumers need to manage the application, middleware, OS, and runtimes, which are deployed in the cloud server and data, which is stored in the cloud storage. Cloud providers, by default, provide security for those applications and data which are managed by cloud consumers. If you store more sensitive data in the cloud, then you can request strict security mechanisms from the cloud provider, which may cost extra. Amazon EC2 is a real example for IaaS, and users can configure and set up a virtual server within one minute [2]. Users are enabled to choose the operation system, type of storage, what kind of file system, network rules for the server and etc.

1.2.2 Platform as a Service (PaaS)

Industries that use the platform solution are empowered to manage the deployed application and storage data. The cloud provider undertakes all types of maintenance in the infrastructure and the OS. They are scaling their infrastructure to cater the application load, and for that cloud, consumers are required to pay a fee. PaaS allows creating and modifying applications. PaaS environments provide consumers with infrastructure as well as fully functioning and development environments for the deployment of consumer applications.

1.2.3 Software as a Service (SaaS)

Cloud providers provide software for consumers to use. It is similar to a multi-tenancy technology adapted to the software, and each vendor/customer utilizes it as a tenant. The consumer can customize the software a little. The SaaS model comprises methodical support of the software, rather than yearly maintenance, fixes and patches, to all subscribers. The SaaS model allows every cloud consumer to take advantage of the latest technological features without the interferences and costs related to software updates and upgrades.

1.2.4 Cloud types

Consumers can choose their cloud usage requirements in terms of infrastructure, applications, OS, and development platforms. Furthermore, consumers can consume the usage of the cloud services permitting the defined service level agreement (SLA) that consists of the desired Quality of Service (QoS) constraints for each service[3]. Those services can be categorized into several aspects using the infrastructure strategy. In that scenario public, private, hybrid, and community clouds come to the picture. Cloud providers sell the public clouds over the internet and cloud consumers can buy those for a pay-per-usage fee. Public clouds can be bought by any person and can be used for any kind of operation using public internet access from any internet service provider. Amazon, Google and Microsoft are some examples of public cloud vendors. There are several advantages of using public clouds such as 100% data and resource availability, easy scaling facility, knowledge of technical experts, any time support, and quick and inexpensive setup. There are several drawbacks to the public clouds too. The main concerns are data security and the privacy of the application of public clouds. Another concern is the reliability of the public cloud. These purchase instances start from the same or different servers as virtual machines so that one server is shared with several cloud consumers. They cannot guarantee that all the servers are allocated to

the same cloud consumer. If the cloud consumer uses the public cloud, then they need to think about the application's sensitive data and the company policies.
For the application, which is using compassionate data, public clouds are not suitable for those purposes. They are using private clouds. In the private cloud, cloud infrastructure is managed by the organization (on-premise) or a trusted third party in off-premise situations. Those clouds can use only for organization users or trusted third-party people and not exposed to public consumers. A private cloud is hosted in the private data center, which belongs to that organization and provides its services only to users inside that organization or its partners. Private clouds give more security for cloud consumers than public clouds. In the private data centers we know how Virtual Machines (VM) are provisioned and how the storages are shared with multiple VMs. So that the people can maintain their own security through private clouds and can protect company policies. In the security and the privacy perspective, private clouds are take advantages and also have several disadvantages of other areas as follows [4],

- In order to maintain the private cloud, we need to maintain the servers, which are quite costly. Need to think about physical server security, backup electricity, networking side and so on.
- Need highly technical people and support people to make the service availability.
- The main disadvantage is the scaling facility and limited resources. In the private clouds cannot increase the server resources on the fly, and also it takes much time to increase the server resources and it is very costly.

Most organizations cannot afford these kinds of costs. As a solution, people are moving to a hybrid solution. In that case, hybrid clouds are come to the world to overcome those limitations. There is no such deployments type in the hybrid clouds and it varies on the organization to organization according to their needs. A hybrid cloud is an arrangement of at least one public cloud and at least one private cloud. A hybrid cloud is usually offered in one of 2 ways such as an organization has a private cloud and starts a partnership with a public cloud provider, or a public cloud provider starts a partnership with an organization that provides private cloud platforms [5].

People who have sensitive data can store the private cloud and can run the application on the public cloud or whole application run on the private cloud and to cater the high traffic, and people can move to public clouds only for that. Preferably, the hybrid cloud method allows the industry to take advantage of the dynamic scalability and cost-effectiveness that a public cloud environment gives without exposing sensitive applications and data to third-party vulnerabilities.

Advantages of hybrid clouds:

- Reduce the organization cost by using public clouds along with the private clouds
- Help to optimize the company cloud infrastructure
- Rapid scaling facility to cater the application load
- Protect application and data privacy
- Extreme improvements in the global business agility
- Ability to influence public clouds and leading to improved opportunities.

Disadvantages of hybrid clouds:

- Introduce the application latency because of the connectivity between the public and the private cloud
- Very complex infrastructure

There are risks related to the security policies across the hybrid cloud environment.

1.3 Monolithic Applications

In ancient time most of the applications deployed in the cloud are monolithic applications. Monolithic applications are a single application that contains all the modules as a single package. That means application modules such as authentication module, business logic and the user interface part are all packaged into a single application. When considering the data layer, most of the monolithic applications typically use one single database to handle all the application data.

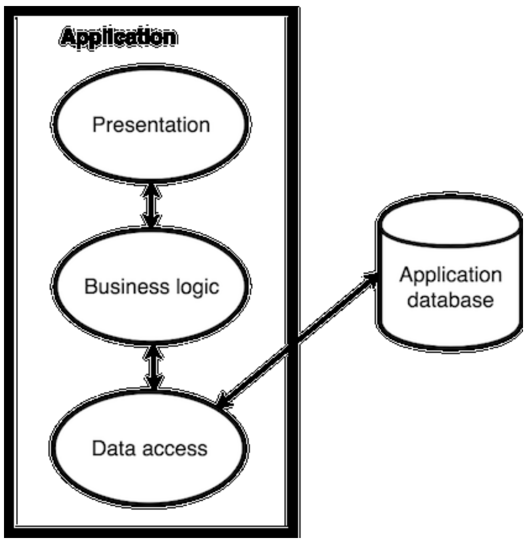


Figure 2: Monolithic application architecture

There are have several advantages and disadvantages of developing monolithic applications.

Advantages –

- In the monolithic application, all the code base is maintained as a single repository so that debugging, patching and maintains of that code base is very easy
- Most of the development IDEs support monolithic application development
- Very easy to scale the monolithic application because it is a single container
- Easy to deploy the monolithic applications.

Disadvantages –

- All the modules are packed into single binary which decreases application performance
- Monolithic application is very complex
- There are several security problems
- Adaptation of the new technology is very difficult
- Cannot maintain the CI/CD pipeline with the monolithic applications

- Not suitable for container orchestration because its takes lot of time to start up the application
- If one module fails in the monolithic application it will cause for whole system failure.

1.4 Microservices

To overcome the limitation on the monolithic applications people move to micro services. This is new technology to the world and most of the organizations are going to move to their monolithic application to the micro service architecture. Micro Service Architecture is a combination of running several independent modules to achieve a common goal. Those modules running as independently and communicate using the APIs by using different protocols such as http, https, JMS, thrift and so on. Micro services can developed using several languages and have several well-known framework also. In the micro service architecture all the modules have their own data layer, and required data will get using API calls.

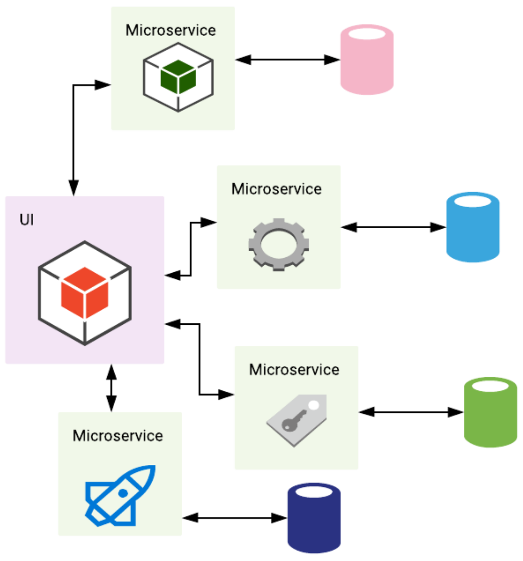


Figure 3: Microservices architecture

There are a number of advantages of using the micro services but also several disadvantages.

Advantages –

- High scalability and the availability.
- If one microservice fault not causes for the whole system downtimes.
- Can perform CI/CD pipeline.
- Container orchestration friendly.
- Parallel development is possible in the microservice architecture.

Disadvantages –

- Maintaining several codebases is very difficult.
- Monitoring the whole system is very difficult because of independent distributed modules.
- Application performance impacts due to network latency between modules.

1.5 Serverless Computing

With the world going with microservices, people think about serverless computing with the concept of the containers and micro services[6]. Serverless computing is referred to as a function as a service in the cloud computing perspective. That is emerging technology and cloud providers facilitate that to the cloud consumers. AWS provides that as lambda services[7] and google provides that as google cloud functions[8]. Serverless architecture is similar to microservice and behaves independent functions on cloud.

Most of the businesses are moving from the monolithic application to the microservice or serverless applications. Business owners are willing to bear the cost of service migration but they are in a risk of complete service migration to microservice or serverless. Business owners always think about the cost of the whole operations and they are looking for revenue to the business within a short period. There are no existing monolithic systems that can work with the microservises as subsystems to satisfy the core system requirements.

There are having several advantages of using serverless functions as applications in the business domain.

- No need to maintain the servers and that cost can be saved by the organization.
- Easy scalability on the services which are running on the serverless.
- Lower cost by considering other cloud services and on premise services.
- Less latency for execution in serverless cloud.
- When considering the development and the deployments, both are very easy and can do with a short time period.

As same as the advantages we see there are some disadvantages also in serverless architecture. One of the many things is serverless architecture is not suitable for massive processing executions. Not suitable means that serverless can process the heavy query, but the cost will be high for that heavy processing. Another thing is debugging on the serverless is a bit hard for the developer because in all the compilation on the code level is done by the cloud provider and didn't have any visibility to the developer about the backend processing. Different cloud providers provide the different implementations and features to serverless architecture on their cloud environment. If the architect chooses the serverless vendor, they also need to consider all of the features and the implementation on that particular vendor. Otherwise, changing the vendor while implementation or after deployment is very hard.

1.6 Problem Description

1.6.1 Goal

The proposed solution will satisfy a cost-effective way to handle the traffic load between server and serverless cloud in a seamless way. This solution will get the server stats such as memory, CPU and disk in the on-premise servers or cloud servers and determine that servers can handle the traffic any more. If that server stats are in a warning state (that can pre-define), this research component can able to reroute the traffic to the particular serverless function which is hosted on any cloud provider. Because of that, traffic is not got failed and can give more user experience without 0%

downtime when traffic spikes. The organization who needs to migrate their monolithic system to the microservices or serverless cloud this research component is much suitable for satisfies that requirement with low risk. Meantime they can run their monolithic system as well as the microservices or serverless cloud application.

1.6.2 Objectives

Objective of this research include the following:

- Analyze the existing cloud provider's services.
- Deeply investigate the cost on maintaining both on premise servers and the cloud servers.
- Prediction of server workloads based on current server statistics.
- Analyze server resource statistics with respect to their capabilities (Load average, memory, and disk) and cost.
- Developing a component to load balance the traffic base of the current server workload.
- Analyze the integration points to integrate newly developed component with the monolithic systems.
- Deeply go through the performance on the microservices.
- Analyze a method for deploy existing microservices on the serverless.

This research shows the cost effective method to load balance between server and serverless cloud. Chapter two talks about critical review of the serverless and monolithic systems.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Nowadays, many types of research are conducted in the field of cloud computing. The goal of this section is to identify the latest research related to cloud computing and what are the problems people are facing in the legacy system. People are trying to move from the monolithic legacy system to cloud solutions. The concept of serverless computing is the biggest trends in the enterprise world. In the literature part researcher is going to analyze this all kind of technologies and have some evaluation about those technologies with respect to performance, scaling and cost.

2.2 Cloud Computing

Over the last years, there has been an intensified interest in the used of cloud computing by organizations for their product deployments. Cloud computing concept induction to the world in the 1960s and from onwards that area is growing very highly. Most of the companies moving to the cloud because of several reasons such as its available for anywhere, easy to use, reliable, maintain cost is low, can get all the services from the cloud provider, and so on. Nowadays, there are have several cloud providers such as Amazon, Google, Microsoft, IBM, Verizon and so on.

2.2.1 Infrastructure as a service (IaaS)

Cloud providers provide the server spaces, memory, CPU, storage, network and other IT infrastructure for the cloud consumers. IaaS cloud provider gives several important fundamental services such as provider takes care of all the IT infrastructure complexities, the provider provides all the infrastructure functionalities, provider guarantees qualified infrastructure services and so on. Cloud consumers can request different storage sizes, different network bandwidths, and different operating systems to match their requirements. The client does not need to worry about resource availability; if you have money, you can buy the IT infrastructure as pay as you go model. If the client uses the IaaS, then client needs to manage the deployed applications, application data and runtime environment. Generally IaaS can be obtained as a public or private infrastructure or a hybrid model. These days AWS EC2, Google cloud, RackSpace Cloud, Linode and GoGrid are the famous IaaS providers in this domain. There are have several frameworks and tools that can help to manage the cloud infrastructure. Those tools giving some extra tools for the enduser to manage the IaaS. OpenStack, Nimbus, OpenNebula and Eucalyptus are some examples for the frameworks.

OpenStack – Openstack is used to manage private and public cloud infrastructure. Developed by the Rackspace and NASA in 2010 and publish it as an open-source project [9]. Openstack framework provide the resource scalability feature, give flexibility and it is compatible for any private cloud managements. Dashboard service (Horizon), authentication service (Keystone), compute service (Nova), block storage service (Cinder), image service (Glance) and object service (swift) are some key services in the openstack [10] [11]. Most of the people use the devstack for creating openstack development environment. Devstack is testing tool and it's not suitable for the production environment.

Nimbus – This consists of the two main parts, such as nimbus platform and the nimbus infrastructure. Nimbus platform provides the tools for the infrastructure management and Nimbus infrastructure compatible for managing Amazon EC2 and Amazon S3 (storage) [12].

OpenNebula – This is also open source IaaS toolkit which is facilitate dynamic resource scaling, live migration of cloud infrastructure, and the making snapshots. It supports various interfaces such as REST base interfaces, open cloud computing interface and so on[13].

Some researchers are researched IaaS base tool to modelling cloud infrastructure. Researcher developed simulator to IaaS, which is named as cloudSim and it is capable for provisioning of a host to a VM, create VMs, destruction of VMs and VM migrations [14]. CloudSim developed using the java language and it gives graphical view of cloud reports to the end user. Using those reports, end user can get the decision about resource utilization.

2.2.2 Platform as a service (PaaS)

Platform as a service (PaaS) is a cloud computing model in which a third-party cloud provider delivers software and hardware tools for those who wanted for product development to use over the internet. A PaaS provider hosts the software and hardware on its own infrastructure. Hence the developer can develop the applications using programming languages and tools supported by the PaaS provider. In the client's perspective, reduce the complexity and responsibility of cloud infrastructure and provide the automatic management to provision resources. Cloud consumers may not be aware of whether provided IT resource is dedicated to a client or shared with other clients. Reduce the responsibility of the runtime environment form the cloud consumers, and they need to manage their application and the data. Do not need to care about how to build, configure, manage and maintain the backend environment. The runtime environment is automatic control such that consumers can focus on their services such as load balancing, fault tolerance, dynamic provisioning and system monitoring. PaaS provides a development and testing platform for running developed applications on the runtime environment so that the development time becomes very short. Microsoft Windows Azure, Google App Engine are some examples of the PaaS providers.

Table 1: Comparison on Azure & Google

Attributes	**Azure**	**Google app engine**
integrity	encryption authentication	encryption authentication
availability	provided by SLA	no SLA, no mention of guaranteed uptime
confidentiality	privacy policy encryption	privacy policy encryption
authentication	single sign-on username & password	single sign-on username & password
system call/ write to file system	denied	denied

Most of the people use the PaaS frameworks for get their job done without using the PaaS on cloud providers. OpenShift, AppScale, Cloudify and Cloud Foundry are some example for the open source PaaS environments. Those PaaS can be deployed on the private, public, hybrid or any community clouds and can choose the development languages like java, ruby and etc. Also, it enables to choose the services like mysql, mongoDB and so on.

OpenShift- Open source framework and that enable to manage, create and deploy the application over the cloud as a PaaS. There is two main parts in the openShift called broker and the node. Broker is responsible for managing all the application activities. Frontend applications use the broker API and interact with the broker. Nodes are the component that host client's applications. OpenShift support to client to choose the resource capacity for their application deployed on the OpenShift PaaS and support to various development languages, various management tools and several database engines.

AppScale – Open source cloud run time system that can run on the cloud servers or virtual machine like XEN or KVM. This execute over the cluster resources and gives the fault tolerance, scalability and availability. AppScale consists of standard three-tier web deployment model with several components such as app controller, app load balancer, app server, database and app scale tools.

Cloud Foundry – This makes the application development and deployment so faster and easier. This is an open source distribution can be used for the private or public cloud environment. Cloud factory supports multiple languages such as java, ruby, Node an also supports several runtimes like grails, groovy, and scala. Also this is support for the rabitmq for the messaging, redis for cache, mongodb for nosql and msql, PostgreSQL for the relational databases.

Cloudify – PaaS contain the 3 main components such as manager, agent and command-line interface client. Manager bootstrapping and managing application is one by the command line interface client, which is written in the python language. Cloudify agent mange the manger's commands with the help of the plugins. Each application has agent to main the application resources in the PaaS cloud. Manager deploys and manages applications on the Cloudify. In this all the operation is done by the REST APIs.

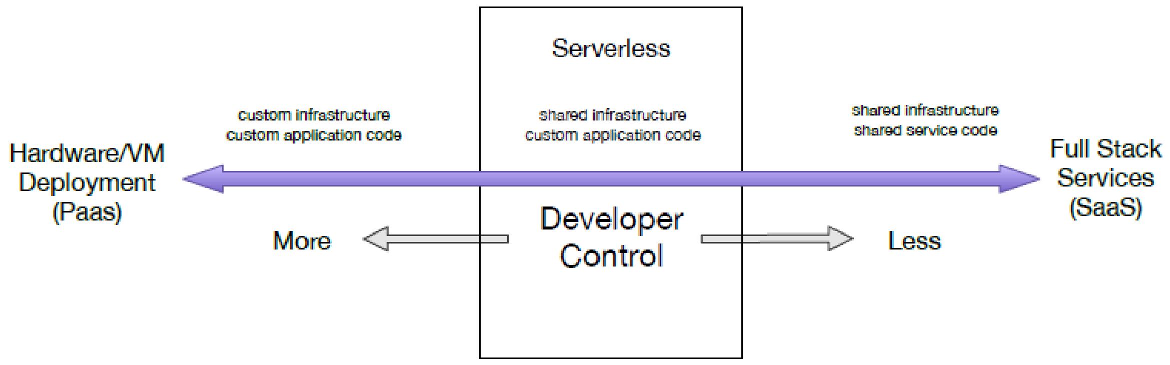
2.2.3 Software as a service (SaaS)

Software as a service is a software program made available on a subscription basis and is centrally hosted by a cloud provider. Instead of installing/running software on-premise, rent the services from a SaaS provider offered online. Historically application service providers gave a similar offerings, but SaaS changed the model to accommodate multi-tenanted architecture, reducing operating costs and increases manageability and maintenance. Cloud consumers can get several advantages such as no installation, monitoring, and maintenance to worry about and no setup time needed, can be accessed from anywhere, can get the service form low cost and so on. In this model, all the application, data and infrastructure is managed by the cloud provider. The main concept of the SaaS is the multi-tenancy. That means getting the software service for the cloud provider and that provider is maintaining service for each tenant.

2.3 Serverless Computing and Microservices

Serverless computing is the trending technology and compelling paradigm for the deployment of cloud-based applications in the modern era. In these days, most of the enterprise applications moving to the container bases applications and the microservice-based applications. While Infrastructure as a Service (IaaS) clouds give users with access to capacious cloud resources, that resource elasticity is control at the virtual machine level, frequently resulting in over-provisioning of cloud resources leading to increased cloud hosting costs, or less provisioning cause for degrade application performance. Serverless platforms, referred to as Function as a Service (FaaS). It reduced cloud hosting costs, fault tolerance, high availability, and dynamic elasticity through programmed provisioning and managing of compute infrastructure to host individual functions known as microservices. Basically, serverless is different than application services hosting with PaaS or IaaS clouds. Applications are modularized into many microservices, which are essentially independent functions. Serverless environments holds operating system containers such as scale microservice and Docker to deploy. Small and powerful code deployment hitch up containers enables incremental, quick scaling of server infrastructure surpassing the elasticity afforded by dynamically scaling instances. Cloud providers can load balance many small containers placing across servers helping to minimize idle server capacity better than with VM placements. Those providers are accountable for creating, destroying, and load balancing requests over container pools. Containers can be re-provisioned and aggregated more rapidly than bulky VMs. To preserve server real estate and energy, cloud providers allow infrastructure to go COLD or de-provisioning containers when service demand is low and freeing infrastructure to be harnessed by others. These efficiencies embrace promise for improved server utilization prominent to workload association and energy savings.

The first serverless platform is the Amazon's AWS Lambda and they define the key dimensions of the serverless functions such as programming model, cost, resource limits, security, deployment and monitoring. In first stage Node.js, Java, Python, and C# are supported for the serverless functions [7]. The platform takes advantage of a vast AWS ecosystem of services and makes it easy to use Lambda functions as event handlers and to provide glue code when composing services. Recently Google and the Microsoft introduced the serversless facility for their cloud consumers [15]. Google Cloud Functions gives basic FaaS to execute serverless functions written in Node.js and other languages in response to HTTP/HTTPS calls or events from some Google Cloud services [16]. Microsoft Azure Functions provides HTTP/HTTPS webhosts and integration with Azure cloud services to execute user delivered functions on using their service. Azure platform supports C#, Python, F#, Node.js, PHP and bash [15].



Serverless
Hardware/VM Deployment (Paas)
custom infrastructure custom application code
shared infrastructure custom application code
shared infrastructure shared service code
Full Stack Services (SaaS)
More
Developer Control
Less

Figure 4: How developer control the serverless cloud

Figure 4 show that in the serverless developer only responsible for the code that is execute in the serverless function. Developer not need to worry about the deployment aspects of the server, fault tolerance of the function and the scaling factor. Number of servers, server capacities are decide based on the code written in the serverless function by the cloud provider[6].

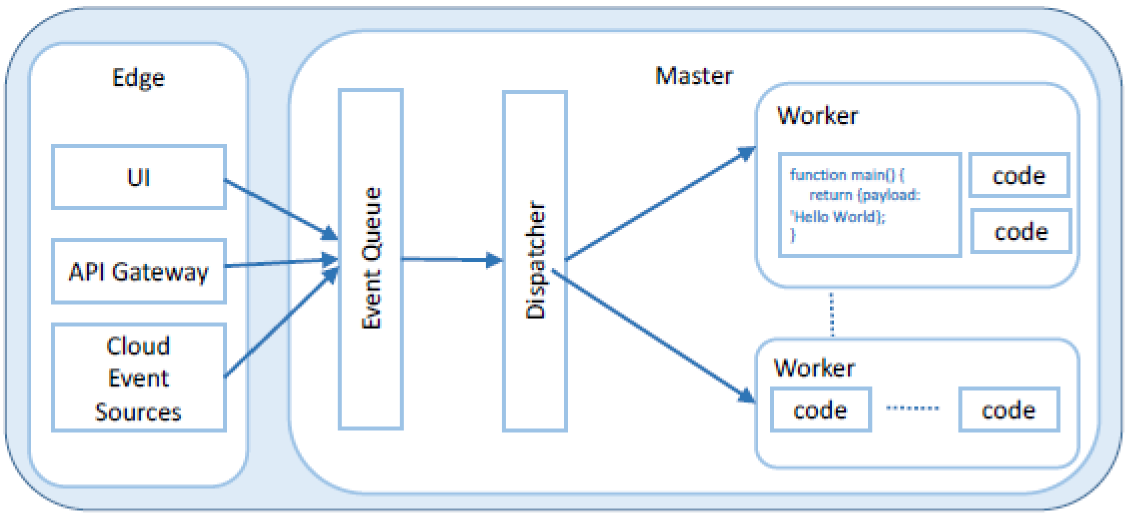


Figure 5: Architecture of serverless platform

Figure 5 show that how serverless worked in the cloud. Serverless platform receive the http traffic and the events from the event source. Then the dispatcher dispatch the event to the correct function to proceed the request. This platform is capable for efficient and quick execution of function and deliver the work in high frequency.
There are have various characteristic that can motivate the cloud consumers to use the serverless clouds.

- Debugging and monitoring – Serverless platform provide the basic debugging capability to the developer to fix the problem such as custom logs, custom traces and etc. Developer can monitor the execution of his function over the serverless cloud.
- Security – Serverless is exposed as multi tenancy but the function execute is done by the isolation and separately by protecting the security.

- Composability – Serverless platform provide the easy way to call a function from another function.
- Deployment – Developer just need to make a source code file or a developer can make his executor as a Docker image to deploy that as a serverless function [17].

Most of the universities done research about the serverless performance by comparing various cloud providers. University of Bloomington did a research about the serverless performance and the serverless function throughput using the data processing tasks [18]. They compare the performance in the memory, CPU and the disk utilization of the serverless function by invocation the requests on sequential and concurrent way. Using that researches can identify the bottleneck on the system. Also researches measure the overall throughput of the http traffic, maximum in/out flow of the storage read and write and the database throughput from the serverless function. They use the three types of triggers to do this performance evaluation.

HTTP Trigger – This means invocation the serverless function via http request with the different format such as JSON, XML, Text and so on. And also http request with different method types such as POST, GET, DELETE, PUT and so on. Non-blocking asynchronous calls are used for the make the concurrent requests for the function.

Database Trigger – Function invoke the database with the insertion, querying, updating and deletion operations. Google functions don't have the database triggers and it supports pub/sub triggers. Then they do their performance analysis using the AWS DynamoDB and the IBM Cloudant.

Object Storage Trigger – Object storage use for the read and write data from the function. Object storages support by the all the cloud providers. Researchers did the comparison on AWS S3, google cloud storage and IBM storage.

Below figure 6 show the throughput of the http triggers on various serverless provider's environments such as AWS, Google, Azure and IBM. For this performance testing they use 500 to 10000 concurrent executions at a given time. As per the test results AWS lambda functions generate 400 TPS in average and it rapidly reach the maximum throughput by executing small number of requests. According to the graph one execution is assigned to a one single server instance which is have shared compute resource serverless functions may take double longer than the sharing CPU.

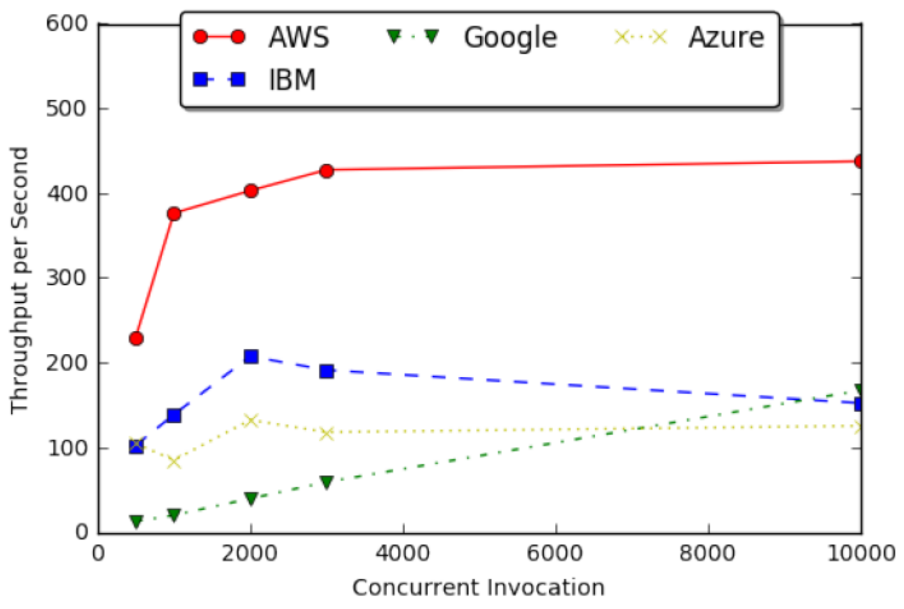


Figure 7: Serverless function throughput on concurrent invocations

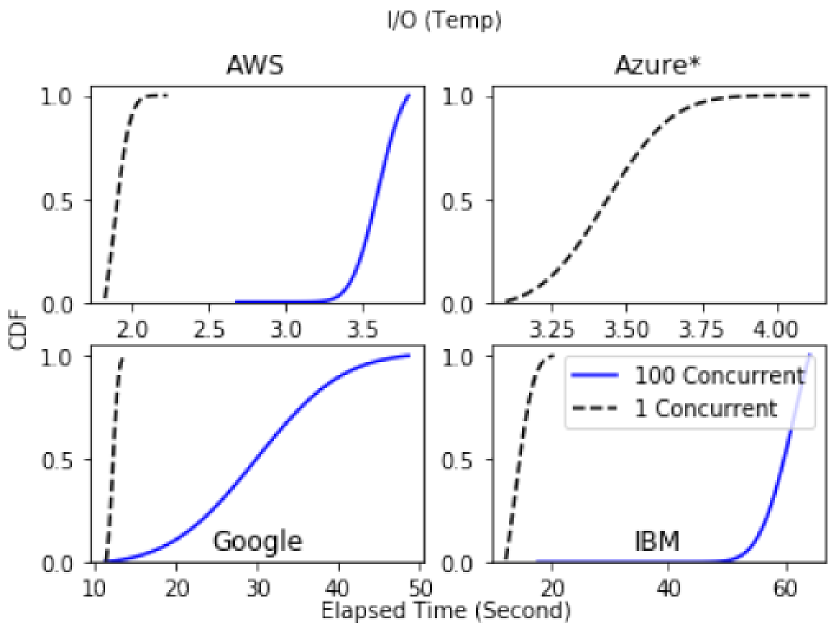


Figure 6: IO overhead while concurrent execution

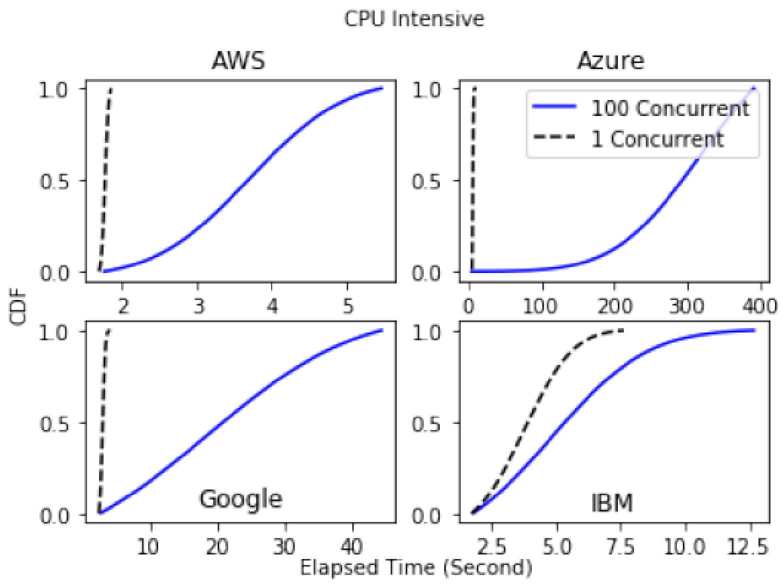
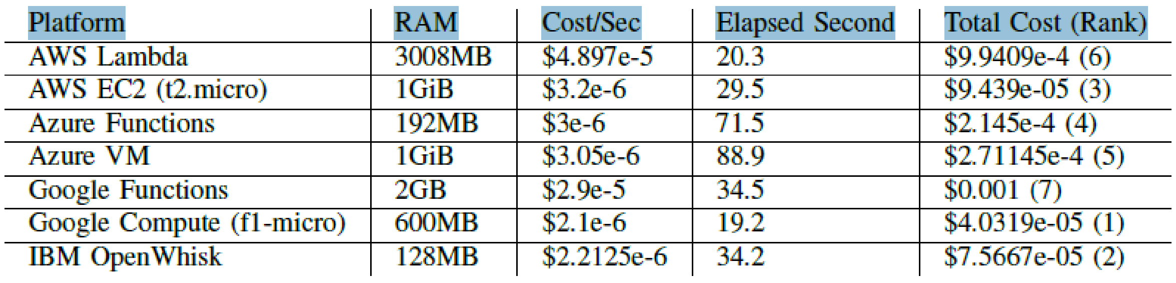


Figure 8: CPU utilization during the concurrent execution

Above figure 7 shows that the how serverless function perform the IO operations (read, write). According to the results over 100 executions, AWS overhead is 91%, Google function overhead is 145% and IBM overhead is 338%. Within the 5 min of time Azure function got failed [18].



Platform	RAM	Cost/Sec	Elapsed Second	Total Cost (Rank)
AWS Lambda	3008MB	$4.897e-5	20.3	$9.9409e-4 (6)
AWS EC2 (t2.micro)	1GiB	$3.2e-6	29.5	$9.439e-05 (3)
Azure Functions	192MB	$3e-6	71.5	$2.145e-4 (4)
Azure VM	1GiB	$3.05e-6	88.9	$2.71145e-4 (5)
Google Functions	2GB	$2.9e-5	34.5	$0.001 (7)
Google Compute (f1-micro)	600MB	$2.1e-6	19.2	$4.0319e-05 (1)
IBM OpenWhisk	128MB	$2.2125e-6	34.2	$7.5667e-05 (2)

Figure 9: Price comparison

Researchers done the price comparison between serverless and the cloud servers (figure 9). Using this comparison we can conclude that the serverless function is much better than the normal cloud servers. And also it gives a better performance than the cloud servers and it makes the developer more easily.

2.4 Workload Load Balancing

In the modern world give the most significant place to distributed computing. With this placement, load balancing comes into the world to equally balance the workload on the distributed servers. Most of the researches give many solutions to the load balancing using the various technologies. The world is moving to the cloud era so that researchers focus on cloud technologies to set up distributed environments and load balance using cloud technologies. Those researches explain the advantages and as well as the drawbacks on the load balancing methods [19]. Load balancing on the distributed environment should not be a static one. It should be an adaptive method of traffic behaviour. However, most of the load balancing algorithms are bounded to some constraints [20]. Most of the researches suggestion is to use AI technologies such as neural networks, evolutionary technologies to build workload balancing algorithms [21]. In the distributed environments communication is happing using the TCP/IP layer. Because of using this mechanism load balancing algorithms face lot of problems. TCP/IP layer is not consistence one. Small changes in the environment will cause some issue on the TCP/IP layer. So it introduces the inconsistency to the load balancing algorithms by introducing the uncertain latency. With this network issue, most of the load balancing algorithms can't perform the expected behaviors to the workload balance. Rest of the chapter show some load balancing algorithms.

2.4.1 Round Robin Method

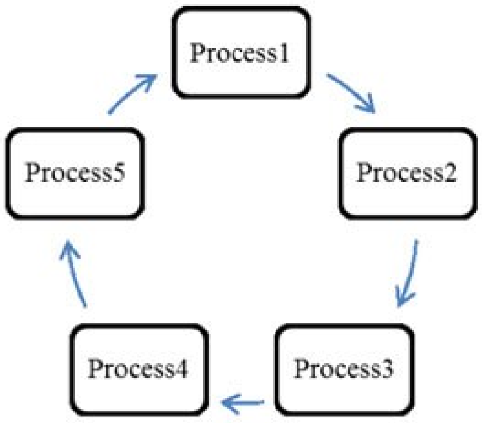


Figure 10: Round Robin method

Round robin stays the most straightforward algorithm by considering the other load balancing algorithms. It has less complexity compared to other algorithms. Because of that, many researches practice RR algorithm to balance the traffic load with the peer systems[22]. It treated as a first come first served manner, which is names as the queue to the incoming traffic and assigns those traffic to the virtual machines by considering the scheduling time. The researcher's commitments on utilizing the RR algorithm for load balancing at instances within the cloud instances as the first request reach the initial instance, and the second request will relocate to the second instance. Besides, the traffic load are decreased to VMs by using this algorithm. Researchers adjust the simple RR process to time scheduling manner. The primary purpose of addling the schedule component to the RR algorithm is to get higher performance during the runtime. The problem of this algorithm is when the instance goes down within the cloud premises that the RR algorithm can't alter on the runtime.

2.4.2 Central Manager Load Balancing Algorithm

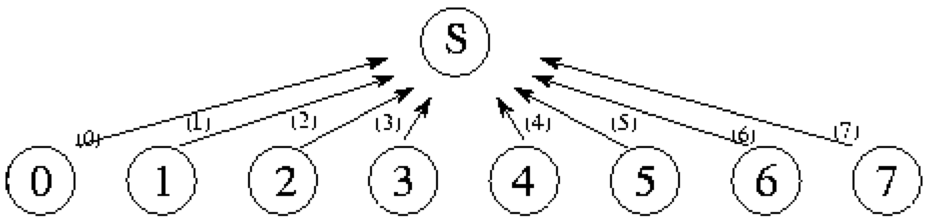


Figure 11: Central manager algorithm

In this algorithm, all the decision is made from the central manager node. The overall algorithm load factor is decided from the connected node and the manager nodes. The central manager node maintains the status of each node workload and gives the jobs to those connected node based on the load status[23]. Every connected node sending its workload to the central manager to keep the status updated. Based on this behavior, sometimes central manger waits for other nodes to complete their assigned worked. Because of that, system performance can be degraded. In that case, the Central Manager algorithms are most fitting for the dynamic environment with less number of nodes.

Table 2: Load balancing algorithm comparison

Parameters	Local Queue	Central Manager	Round Robin
Resource Utilization	Less	Less	Less
Centralized or Decentralized	Decentralized	Centralized	Decentralized
Stability	No	Yes	Yes
Fault Tolerant	No	Yes	Yes
Dynamic or Static	Dynamic	Static	Static
Overload Rejection	Yes	No	No

Cluster is the new technology in the load balancing. With this technology, people can share the data inside the cluster for load balancing. But when come to the adaptive balancing it is huge computing power to process those data and get the decision. That will cause for the application performance and introduce a lot of complexity to the application.

*2.4.*3 Self-Adaptive Load Balancing Algorithm

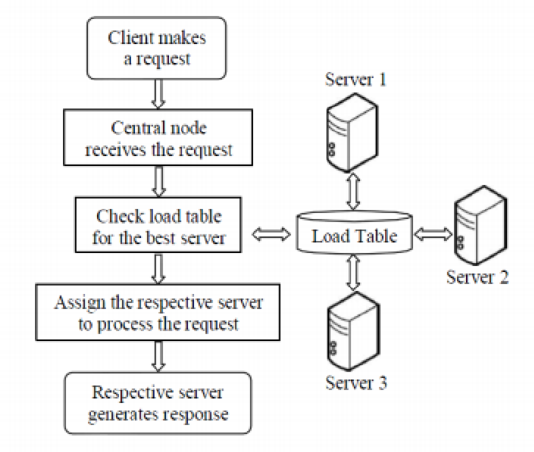

Client makes a request
Central node receives the request
Check load table for the best server
Assign the respective server to process the request
Respective server generates response
Server 1
Load Table
Server 2
Server 3

Figure 12: Work flow of self-adaptive load balancing

Most of the algorithm does not consider the server response time, server load average and several other request details. They only consider the traffic hit to the algorithm. Round robin load balancing algorithm and the weighted load-balancing algorithm is some example for simply algorithmic module. Self-adaptive algorithms are can mark as dynamic algorithms which are better than static algorithms. Self-adaptive load balancing system mainly includes two main parts. First one is monitoring the load statistics of servers and the second one is assigning the request to the relevant servers. In this researchers server CPU load is determining as per the below rule [24].

$$\textbf{Load} = \begin{cases} \text{idle} - \text{cpu\_u} < 1\% \\ \text{high} - (\text{mem\_u} > 85\%)\text{or } (\text{cpu\_u is high}) \\ \quad \text{or } (\text{n\_u is high}) \\ \text{low} - (\text{cpu\_u is low and n\_u is low})\text{or} \\ \quad (\text{n\_u is low}) \\ \text{normal} - \text{ otherwise} \end{cases}$$

Load n – Load value of the node n.

cpu_u n – Average of CPU utilization over a period of 5 seconds.

mem_u n - Average of memory utilization over a period of 5 seconds.

n_u n - Average network traffic over a period of 5 seconds.

Equation 1: Load calculation

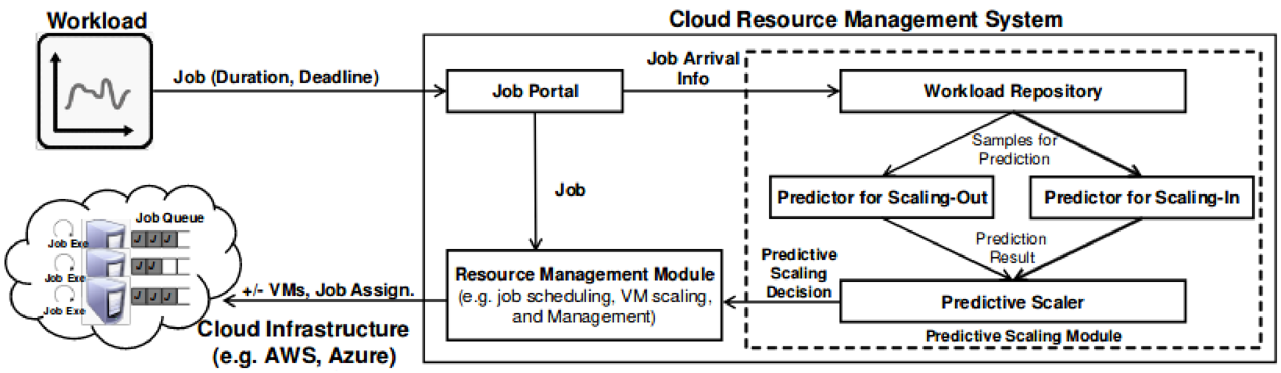
Some algorithm developed to consider the server statistics and get the decision using those variables. The self-Adaptive load-balancing algorithm is similar to that. Centralize server collected all server statistics and make the decision that which server has the low statistics and traffic routed to that server. According to this research, all the servers help to the main algorithm to get the decision. It is a collective centralized work management.

2.5 Workload Prediction and Resource Allocation

Nowadays resource provisioning is one of the key challenge in cloud computing by considering the cost and the actual workload. Most of the cases engineers do the over provisioning and that cause for heavy costing for the business. Some of the organization allocation the less resources for the server and finally it will cause for the live traffic interruptions and the quality of service level. To achieve the proper resource allocation people want know the application and the traffic patterns. It is not easy to the determine the traffic pattern of the application because throughout the year users get on board to the services so we can’t predict the end user behaviors. To overcome these kind of things people need to proactive resource allocations. Below are the challenges for proactive resource allocations in the cloud environment [25],

- Implementing the server resource prediction models for proactive auto scaling in the cloud environment and in order to do that application need to understand the fluctuation in the current traffic and the workload.
- Implementing the application that can optimal allocation the CPU and the sever memory to incoming traffic.
- Encompassing the prediction algorithms, which are allowing for single dimension of the server application, to also examine all dimensions of the applications.
- Avoiding bottleneck workload by correlating the server resources.
- Manipulating the resource provisioning scheme.
- Implementing the approaches for provisioning the resources to cloud applications that reduce the overall costing
- Designing the failover mechanism

The workload prediction model can be categorized into 4 classes [26]. Those are Naive workload predictors, Regression-based workload predictors, Temporal (Time-Series)-based Workload Predictors and Non-temporal Workload Predictors. Native workload predictors consist with the mean and the resent mean based algorithms. Regression-based workload predicators consist with the linear (1-degree) or polynomial (2 or more degrees) models. Temporal (Time-Series)-based Workload Predictors exist numerous temporal (time-series) approaches for the upcoming workload prediction since these predictors are commonly used to analyze workload patterns for cloud computing. Non-temporal Workload Predictors are the predictors that have not been applied to the cloud resource scaling before. They have delivered accurate prediction results within a deterministic amount of time. Researchers consider numerous non-temporal methods to forecast the next job arrival time.



Workload
Job (Duration, Deadline)
Cloud Resource Management System
Job Portal
Job Arrival Info
Workload Repository
Samples for Prediction
Job
Predictor for Scaling-Out
Predictor for Scaling-In
Prediction Result
Predictive Scaling Decision
Resource Management Module (e.g. job scheduling, VM scaling, and Management)
Predictive Scaler
Predictive Scaling Module
Job Queue
Job Exe
Job Exe
Job Exe
+/- VMs, Job Assign.
Cloud Infrastructure (e.g. AWS, Azure)

Figure 13: How work perdition happen

Above image show the how predicates are used in the actual environment for allocate the resources to the cloud resources.

There are have some challenges perdition approaches;

- Examine of the future behavior pattern of the application in different perspectives such as TPS, promotion campaigns, service level agreements is a challenging task.
- In the perdition algorithm should be good in time complexity.
- In the initial state data generality is more importance to make the flexible perdition model. But long term data will cause for effectiveness of the model.

Finding the relation between the attributes on the perdition model is very hard on the cloud environment.

Some of the researchers bring the rule base resource allocation methods, which are considered the server memory, CPU and storage. Researchers prove that the resource allocation rate should be greater than the resource request rate from their research [27].

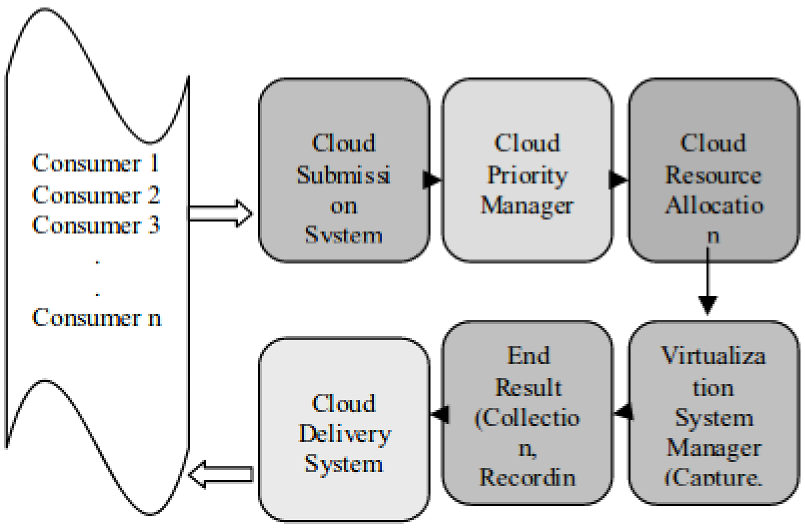


Figure 14: Components of the rule based resource allocation

Figure 14 shows that the major components of the rule-based resource allocation system are cloud priority manager, cloud resource allocation, virtualization system manager, and result collection. From the research results, they show that cloud resource allocation is very efficient using the rule-based allocation algorithm.

Grid and Service Computing Technology Lab did research related to the effective resource allocation problem based on real-time information on workload and performance feedback of running services in the production environment [28].

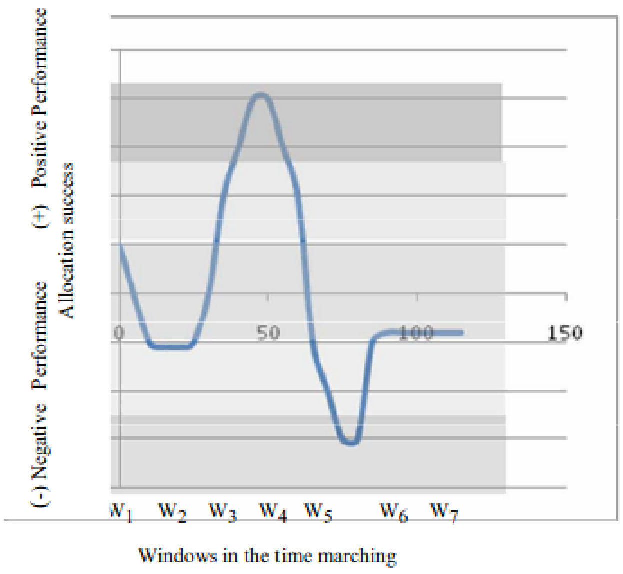


Figure 15: Results graph after resource allocation

Researches proposed a stochastic model for the server resources on the virtualized environment and scheduling heuristics algorithms and resource allocation with the SLA (Service Level Agreement) constraints. In this research, researchers marked front face resource (external LB, VM server) as a performance feedback mechanism to the source because that is the place all the traffic come into the system. That is the better place to get the details regarding the current workload and the past workloads. The developed algorithm interact with the server resources by using the closed-loop policies to set a nontoxic performance level in addition it considers the synchronized optimization of several SLA requirements together. This method address the address the hotspots problem between multiple VMs through migrations [29].

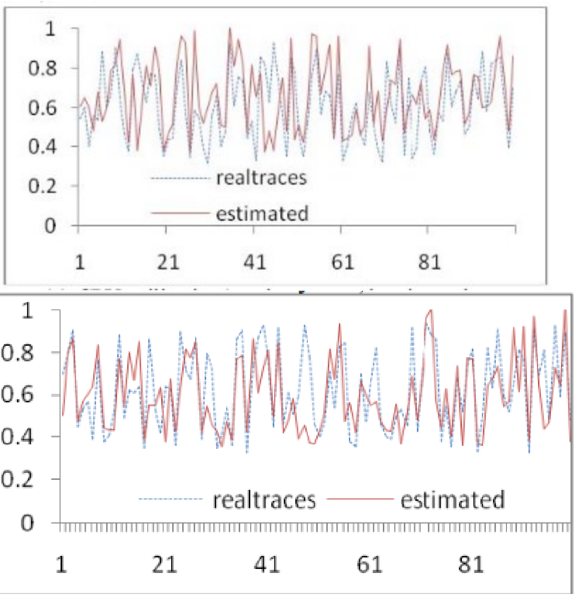


Figure 16: CPU load average's difference

Figure 16 shows how that developed algorithm works with the actual environment. According to the above graph estimation level and the real stats are somewhat similar.

2.1.5 Auto Scaling

In the computing world, there are having two types of scaling. They are horizontal scaling and vertical scaling. Vertical scaling meaning that add the server resources (RAM, Cores, and Disks) to the existing server. Horizontal scaling means adding another server which can run parallel to existing resources.

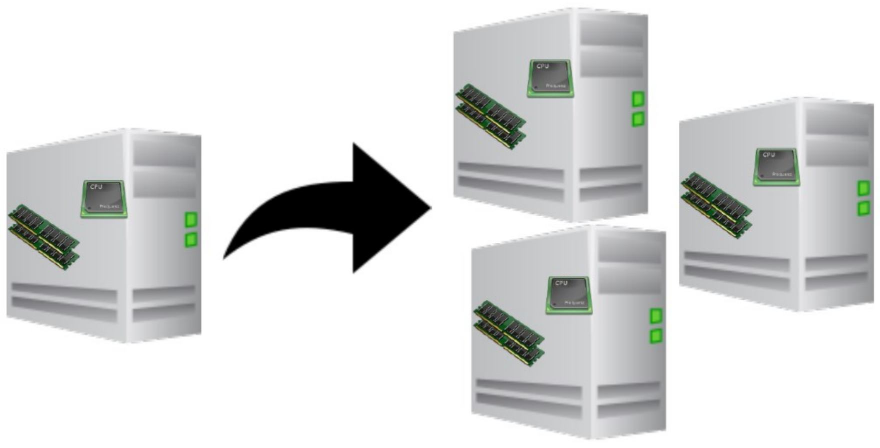


Figure 17: Horizontal scaling

Figure 17 shows horizontal scaling on the servers. This scaling type is having advantages and disadvantages.

Advantages –

- Lower cost by comparing to the vertical scaling.
- Increase high availability.
- Introduce the fault tolerance to the application level.
- Ability to scale our much more by placing the load balancer in front of the servers.

Disadvantages –

- Most of the software does not support horizontal scaling.
- Higher cost for server managing the maintenance.
- The software needs to handle the logic across multiple environments.

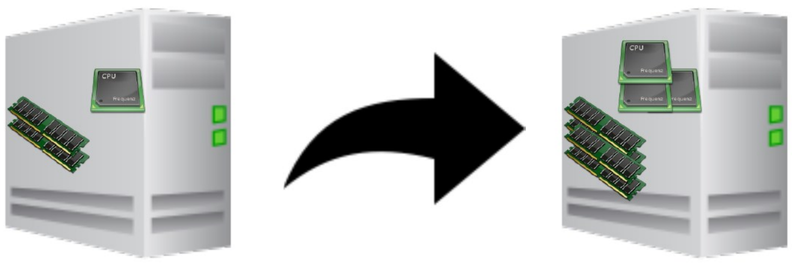


Figure 18: Vertical scaling

Figure 18 shows that the vertical scaling on the server and from the vertical scaling we have advantages and the disadvantages also.

Advantages –

- Most of the monolithic and microservices systems are support for this type of scaling
- Very easy to scaling
- Most of the virtualization engines support for vertical scaling

Disadvantages –

- There is limitation to a certain limit to scale.
- Required very big amount of costing.
- High risk of failing the hardware.
- Having a low availability.

Cloud providers also provide the auto-scaling facility to their cloud consumers. In the AWS cloud, they are providing this feature and consumers can set up according to the defined user policies. There is another service called as predictive scaling on the AWS uses the machine learning algorithms and detect changes in daily and weekly patterns, then automatically adjusting their forecasts.

When the cloud consumer used the AWS auto-scaling that person can earn several benefits such as,

- Can setup quickly
- Can make smart scalable decisions
- Only cost for scale up resources

Google Cloud also supports the auto-scaling services for their VMs. Alibaba Cloud also provides the same mechanism as Google cloud to scaling. These all scaling are under the horizontal scaling and load-based scaling and the time bases scaling.

CHAPTER 3
METHODOLOGY

3.1 Introduction

This section will introduce the method of the dynamic load balancing between the serverless architecture and the servers. This strategy will help to build a hybrid model between monolithic applications and microservices architecture. This model helped to seamless migration to the between monolithic and the microservices architecture.
In order to implement this research project, it is mandatory to do a qualitative and quantitative research analysis on the different research areas and different technologies. Such as all the aspects of cloud computing, serverless computing research areas, microservices and its frameworks, load balancing mechanism and scheduling tasks. Chapter 2 constructs a literature review about similar researches conducted by other people in this field of study. In that chapter, we critically evaluate how those researches use emerging technologies to solve the problem and make their research successful.
Those researches provide evidence that using the latest technologies will be helpful for the success of this research. We will utilize those research experience for the betterment of this research. Moreover, we will need to come up with new methodologies to build up this component.

3.2 Converting Monolithic application to Microservices

Monolithic architecture is a traditional method used by organizations to build their application. Most of the monolithic applications are single-tier applications and all the components combined into a single program. Older days, people are moved to the monolithic application because of that is very easy to build and can deliver the application within a short time period. Older day's software engineers focus on the web application, which is built for desktops or laptops. So that those applications are browser oriented applications. After the technology changed, every person is trying to access to the system through the smart phones such as tablets, phones, ipads and so on.

With the many devices come into the picture, developers think that to expose services via APIs to access from any components. Then they pretended to developed small components as a service by exposing the service via API. The easiest way to expose APIs is by building the program as microservices. Then most of the new enterprise applications were build as microservices.

There are three main strategies to move monolithic applications to the microservices. It's not a straight forward task to get done with months. People can develop new features as microservices can work with the existing monolithic application. However, this approach takes a long time to move the whole of the system. Another strategy is to separate the application front end part from the backend part. A monolithic application is tightly couple application so that separating UI modules and the backend modules is not an easy task. Another complex strategy is to write business logic as a service on the monolithic system. So those three strategies are very costly and it takes a lot of time to move from the monolithic to the microservices environment.

In this research presents the hybrid way, which is shown in figure 19 to handle the monolithic and the microservice as one platform. This is the combination of the above three strategies. From this strategy, the business owners can evaluate both systems as performance-wise, quality-wise and as well as cost-wise. This will help to bring the enterprise system into the latest technology in the modern world.

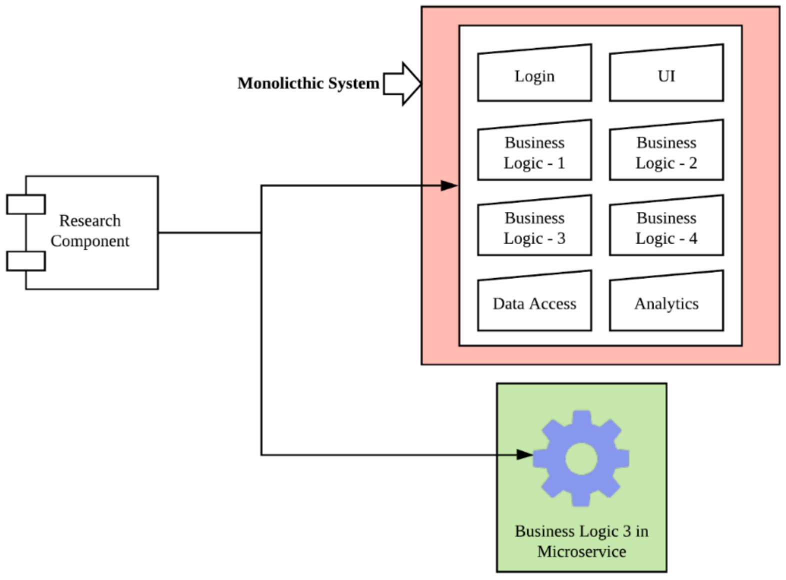


Figure 19: Hybrid approach

3.3 Converting Microservices into serverless

When building microservices architecture, developers need to think about what are the services/modules in the whole application. As a business-wise, also we need to think about what are the business logic we can develop as separate components to reduce the complexity. After defining the services, then need to expose those service via APIs. Because each service has a dependency on each service. So one service needs to access other service responses to complete the task.

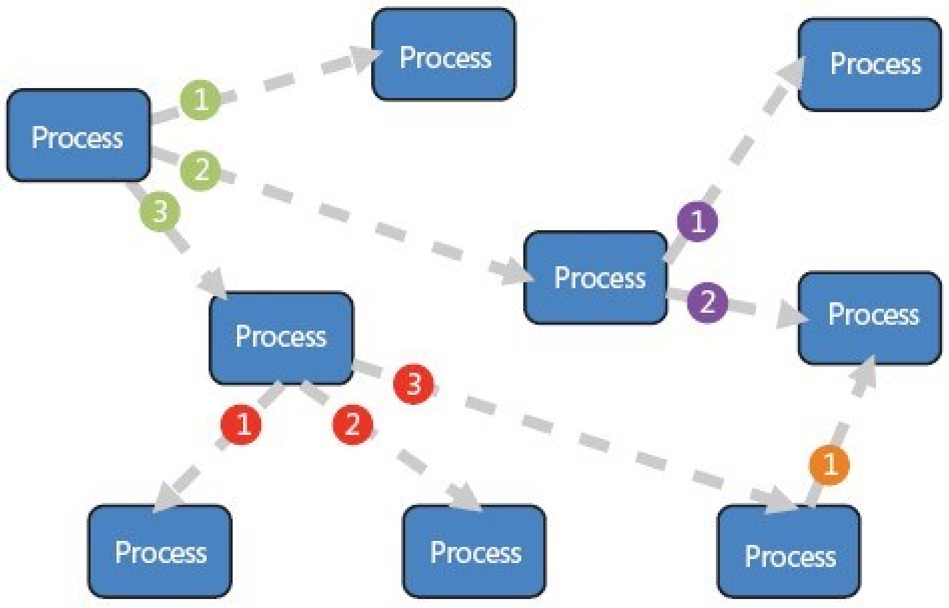


Figure 20: Microservices communication architecture

As per the figure 20, those services cannot run on the individual to perform the job. But they can run separately and produce the assigned job. When designing the microservices architecture, we need to follow a Conway's Law anti-pattern. Conway's law elaborates the system architecture as same as the organizational structure. That structure is very hieratical and tightly coupled with the entities and those model suitable for the monolithic systems. Because of that, when designing the microservices, developers need to follow the Conway's Law anti-pattern. When developing the microservices architecture, developers need to think about service quality and the agreed SLAs to the relevant parties. Microservices can be developed using several languages Java, Ruby, Python and so on. But in this research only focus on the Java implementation because most of the industry using the Java languages and it's frameworks for implement microservices. In the java, there are have several well-known frameworks for microservices such as spring boot, DropWizard and Jersey.

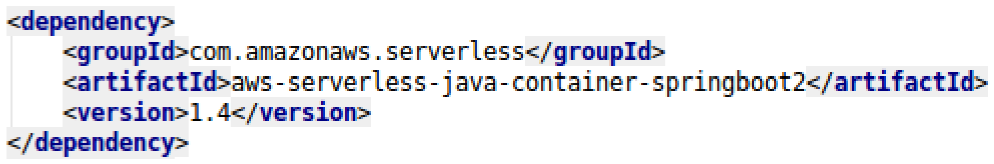
Spring-Boot – This is the most famous framework used to develop the microservices using Java. It has a consistent infrastructure that is appropriate for any kind of apps such as big data, security and etc. That framework facilitates the 3 layer architecture modules to build the microservices efficiently and reliably.

DropWizard - This microservices framework facilitates to develop very industrious RESTful web services. This DropWizard framework is an open-source and holds tested in practice Java libraries. This reduces the developer's time and efforts when building and testing web services.

Jersey - Formed on JAX-RS specification, Jersey is known for high speed, simple routing, clear documentation. Jersey is most famous as a web server among the developers.

We can't deploy the microservices into the serverless as it is. Because in the serverless, they need some of the handlers, some cloud tokens to identify the API calls and so on. So we need to convert existing microservices to the serverless model which can deploy the services as functions. In this research, we talk about the Spring-boot services and the AWS lambda services. The application should need to based on the spring-boot-starter-web. Developed code needs to include a StreamLambdaHandler class, which is the main entry point for AWS Lambda and the rest of the Application class can define as a basic Spring Boot application.

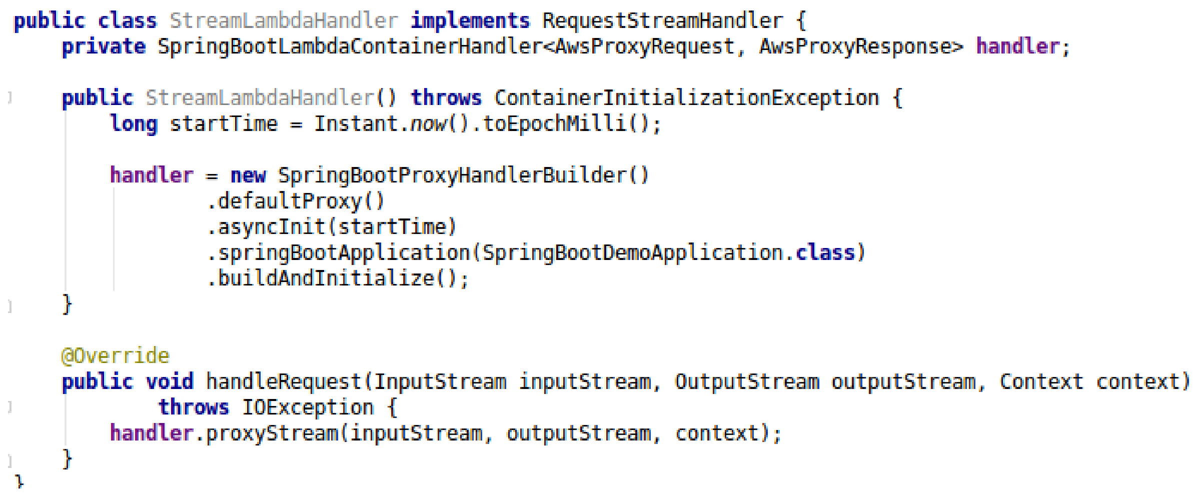
To implement those classes first, you need to import the following artifacts.



```
<dependency>
    <groupId>com.amazonaws.serverless</groupId>
    <artifactId>aws-serverless-java-container-springboot2</artifactId>
    <version>1.4</version>
</dependency>
```

Figure 21: Java dependency for AWS serverless

This dependency will automatically import the aws-serverless-java-container-core and aws-lambda-java-core libraries. Dependency injection with Spring boot can have a major impact on serverless function's cold start time. As the entry points in the application level, needs to declare a new class and need to implement the Lambda's RequestStreamHandler interface. If your architecture configured AWS API Gateway with a proxy integration, that automatically built-in POJOs AwsProxyRequest and AwsProxyResponse in the integration layer. The next step is to write the container handler object in the application class. The library representations the efficacy static method that can configure a SpringBootLambdaContainerHandler object for AWS proxy events. That method receives a class annotated with Spring Boot's @SpringBootApplication. And the object should be declared as a class property and be static. Because of this, Lambda will re-use the instance for subsequent requests and that will bring more performance to the lambda function. The handleRequest method of the class can use the handler object that declared in the previous step to send requests to the Spring application.



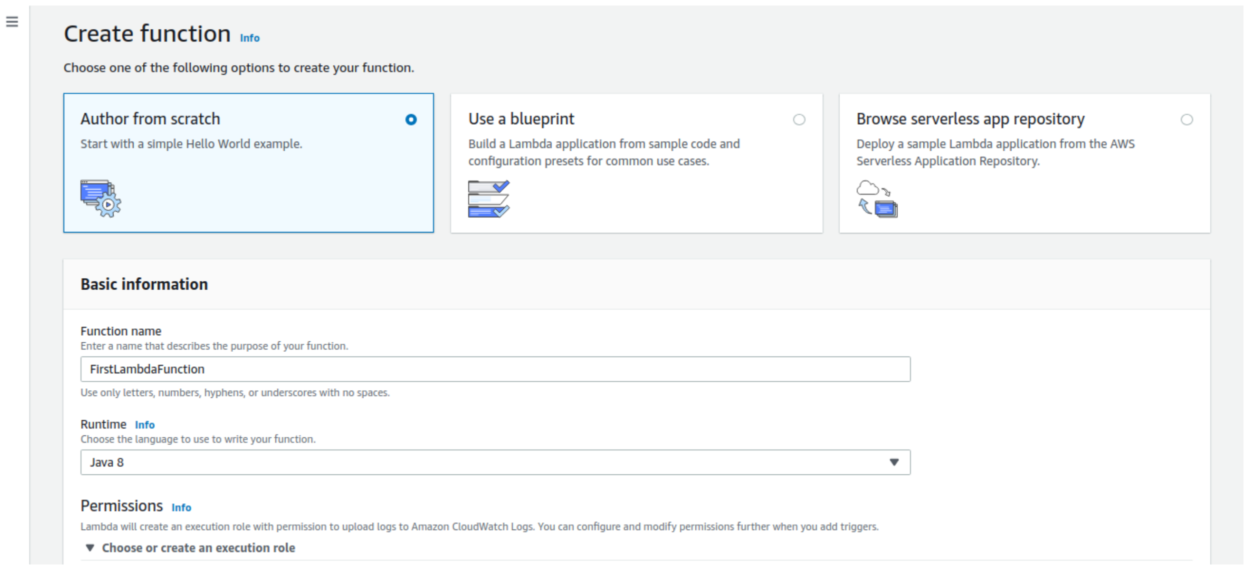
```
public class StreamLambdaHandler implements RequestStreamHandler {
    private SpringBootLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;

    public StreamLambdaHandler() throws ContainerInitializationException {
        long startTime = Instant.now().toEpochMilli();

        handler = new SpringBootProxyHandlerBuilder()
                .defaultProxy()
                .asyncInit(startTime)
                .springBootApplication(SpringBootDemoApplication.class)
                .buildAndInitialize();
    }

    @Override
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
            throws IOException {
        handler.proxyStream(inputStream, outputStream, context);
    }
}
```

Figure 22: Java wrapper for microservices

From wrapping the exciting spring boot microservices with this interface developer’s can easily deploy the application on the serverless function at the AWS lambda.

3.4 Serverless (AWS)

Serverless is a native cloud architecture for the application and can move all the responsibilities to the cloud providers. The only developers need to consider the code and the business logic on that function. Serverless eliminates the people to think about infra management, resource provisioning, operation system maintenance, patching, security and so on. Serverless manages all computations, maintains the high availability and all of the infra related things manage by the serverless provider. If you use the serverless, you can develop modern applications into enterprise-level with increased quickness and lower total cost of manage and ownership. This minimized the overhead of the developer's dev time and energy that can be spent on developing great products which scale and that are reliable.
Lambda is serverless computing service on the AWS and developers can write the code using Java, python, Ruby, Go and the C# languages. This research talk about related to the java development on the AWS lambda.
In the lambda you can create a function from scratch or can use the temples which is provided by the cloud provider and also can choose the language.


Create function Info

Choose one of the following options to create your function.

Author from scratch ●
Start with a simple Hello World example.

Use a blueprint ○
Build a Lambda application from sample code and configuration presets for common use cases.

Browse serverless app repository ○
Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

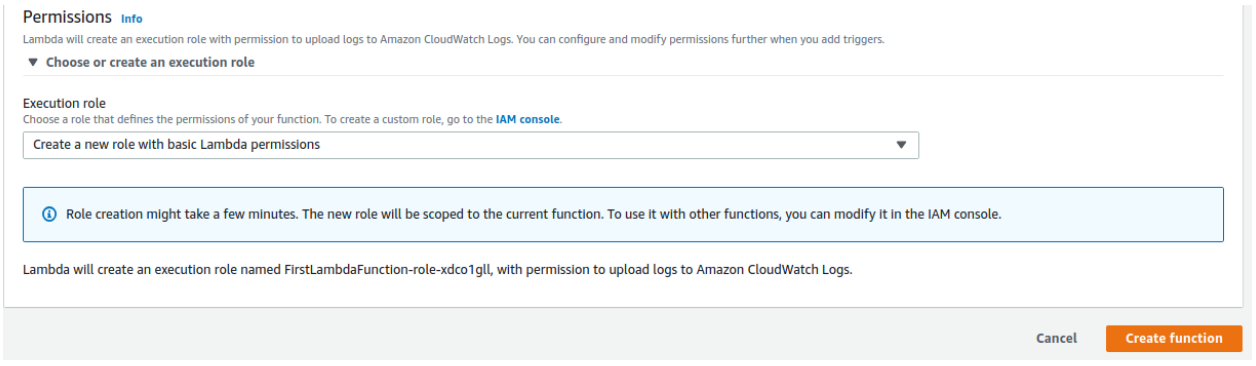
Function name
Enter a name that describes the purpose of your function.
FirstLambdaFunction
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.
Java 8

Permissions Info
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▼ Choose or create an execution role

Figure 23: Creating Lambda function

Afterwards, developer can enable the permissions for the lambda functions. From this, it secures the business functions and data from the outsiders. We can assign role bases permission to each function on the AWS lambda.


Permissions Info

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ Choose or create an execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

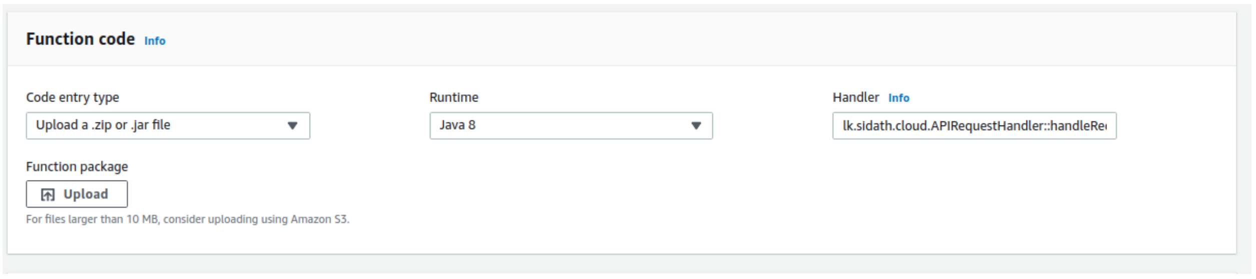
Create a new role with basic Lambda permissions

Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

Lambda will create an execution role named FirstLambdaFunction-role-xdco1gll, with permission to upload logs to Amazon CloudWatch Logs.

Figure 24: Creating roles on Lambda

After entering the whole configuration and the permissions developer can upload the application file to the serverless.


Function code Info

Code entry type: Upload a .zip or .jar file

Runtime: Java 8

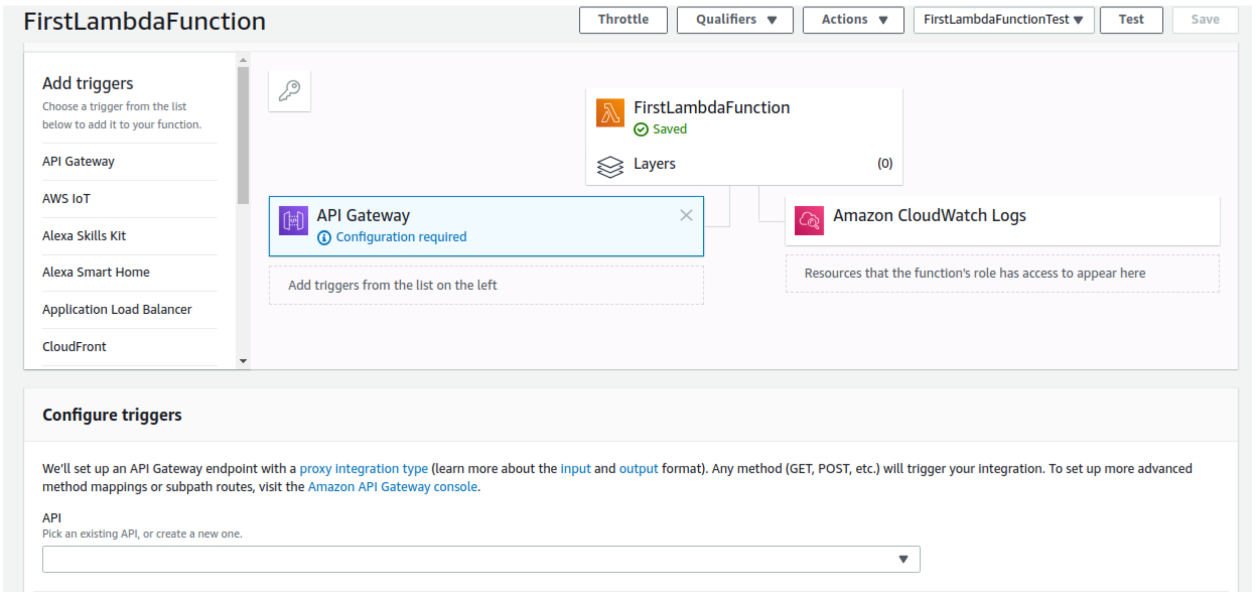
Handler Info: lk.sidath.cloud.APIRequestHandler::handleRe

Function package

For files larger than 10 MB, consider uploading using Amazon S3.

Figure 25: Uploading jars to Lambda

After uploading the code, it is ready for the invocation. People can directly invoke the lambda function after this. But when it comes to the integration, we need to front the gateway in front of the business logic to manage and monetize the of APIs. In this configuration, we can define the API as a new API and can provide the security with the API key. After creating a new API you can see the API details with the API key.


FirstLambdaFunction

Add triggers
Choose a trigger from the list below to add it to your function.

API Gateway
AWS IoT
Alexa Skills Kit
Alexa Smart Home
Application Load Balancer
CloudFront

FirstLambdaFunction
Saved
Layers (0)

API Gateway
Configuration required

Add triggers from the list on the left

Amazon CloudWatch Logs

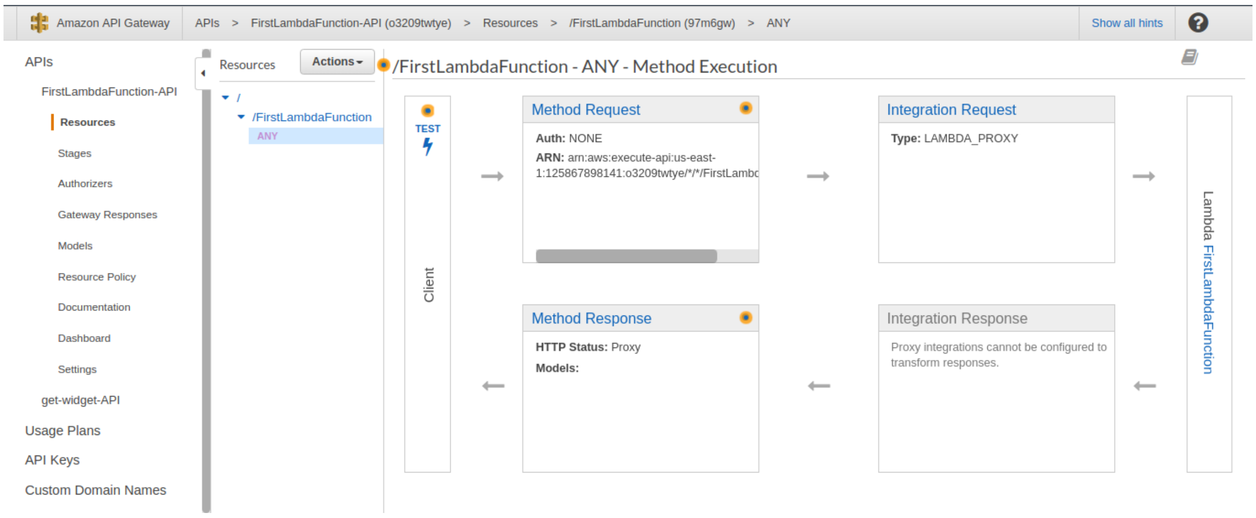
Resources that the function's role has access to appear here

Configure triggers

We'll set up an API Gateway endpoint with a proxy integration type (learn more about the input and output format). Any method (GET, POST, etc.) will trigger your integration. To set up more advanced method mappings or subpath routes, visit the Amazon API Gateway console.

API
Pick an existing API, or create a new one.

Figure 26: Adding API gateway


Amazon API Gateway > APIs > FirstLambdaFunction-API (o3209twtye) > Resources > /FirstLambdaFunction (97m6gw) > ANY

/FirstLambdaFunction - ANY - Method Execution

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:125867898141:o3209twtye/*/*/FirstLamb

Integration Request
Type: LAMBDA_PROXY

Method Response
HTTP Status: Proxy
Models:

Integration Response
Proxy integrations cannot be configured to transform responses.

Figure 27: Integration on Lambda

Above figures 25 and 26 show the integration architecture of the serverless (Lambda) deployment on the AWS. From this architecture all the logs and the all the monitoring is managed by the AWS.

3.5 Server Status Monitoring – NRPE

Nagios is a very powerful monitoring system across both on-premise and the cloud environments. Most Nagios used for the monitors the environment infrastructures rather than the service calls. Nagios architecture is intensely predisposed by a separation of concerns approach that differentiates the monitoring infrastructure from the analyses that monitor the resources. This method is taken to the design of a plugin oriented framework. So the best plugin is Nagios Remote Plugin Executor (NRPE). There are two modes on the NRPE such as server and client mode. The server module is need install on the every cloud or on-premise resource.

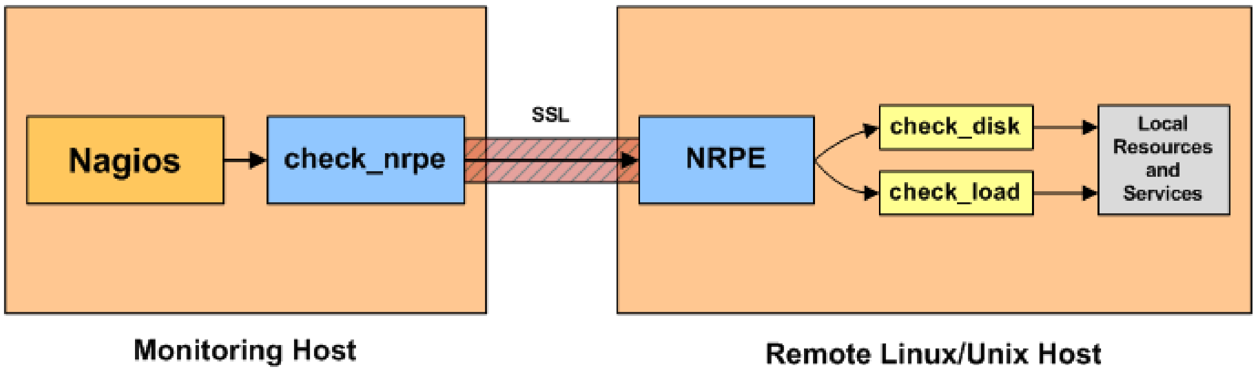
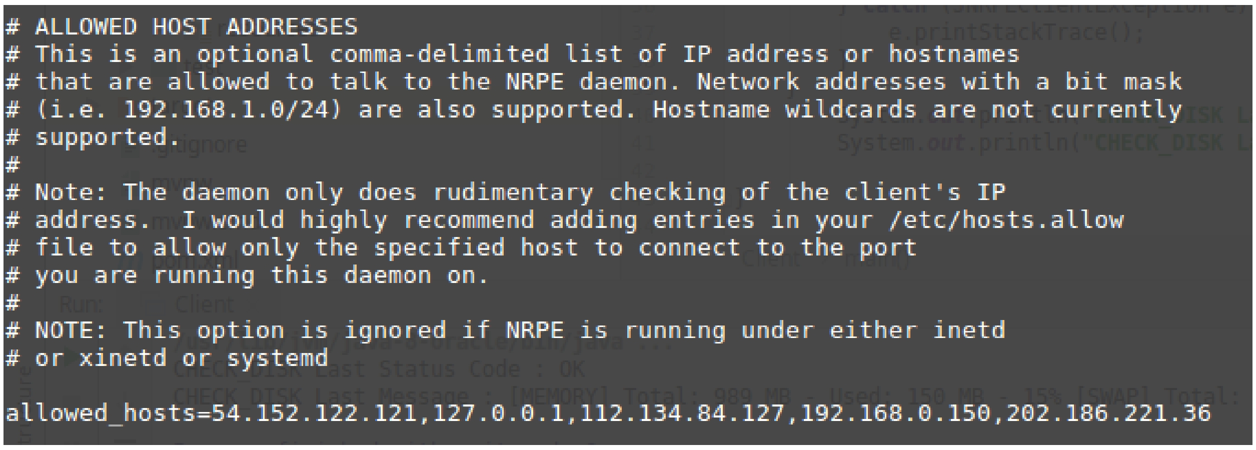


Figure 28: NRPE architecture

Figure 28 shows the NRPE architecture that joins the constancy given by the use of a standard protocol and by the integrated development of the NRPE plugins with the flexibility of the plugin mechanism, which allows the continuous overview and improvement of plugins. The NRPE server gives remote access via IP whitelisting on the configurations to a number of servers which are designed to perform hardware and software checks of the remote host. Finger 29 shows the configuration on the nrpe.cfg file. The monitoring servers control the execution of remote plugins hosted by NRPE servers, and gets data with a secure connection via a standard protocol like SSL.



```
# ALLOWED HOST ADDRESSES
# This is an optional comma-delimited list of IP address or hostnames
# that are allowed to talk to the NRPE daemon. Network addresses with a bit mask
# (i.e. 192.168.1.0/24) are also supported. Hostname wildcards are not currently
# supported.
#
# Note: The daemon only does rudimentary checking of the client's IP
# address.  I would highly recommend adding entries in your /etc/hosts.allow
# file to allow only the specified host to connect to the port
# you are running this daemon on.
#
# NOTE: This option is ignored if NRPE is running under either inetd
# or xinetd or systemd

allowed_hosts=54.152.122.121,127.0.0.1,112.134.84.127,192.168.0.150,202.186.221.36
```

Figure 29: NRPE configurations on security

In this research implementation, used the NRPE plugin in each server to get the data on reach remote servers.

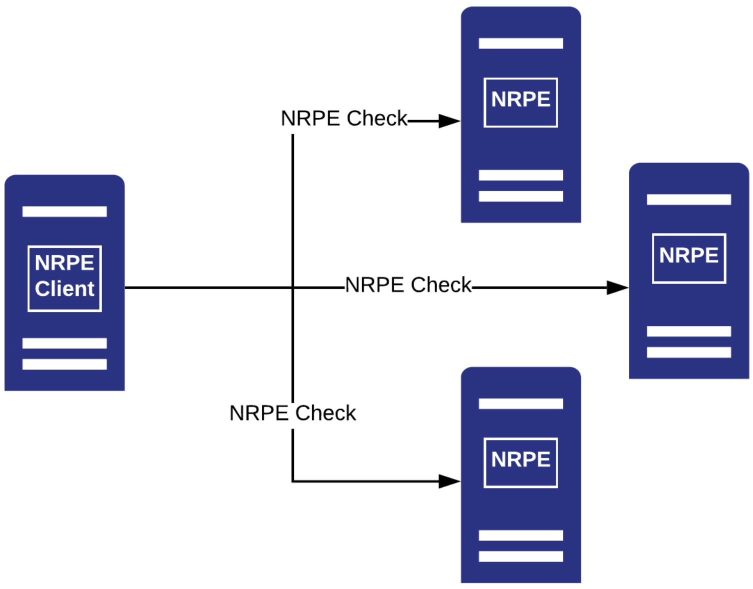
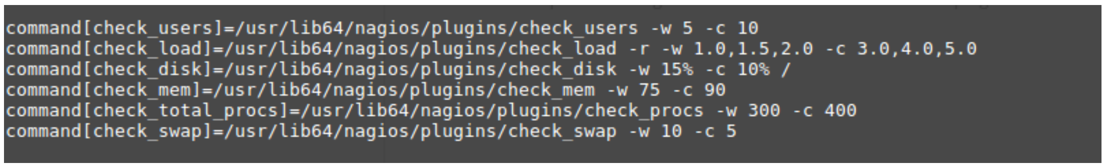


Figure 30: NRPE implantation

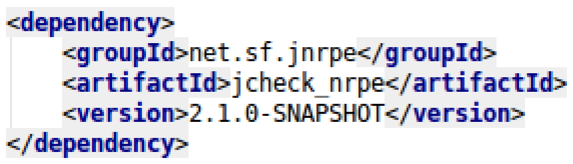
According to the figure 30, there is having a Java-based NREP client that is capable to get the data from the installed NRPE plugins on the remote servers.



```
command[check_users]=/usr/lib64/nagios/plugins/check_users -w 5 -c 10
command[check_load]=/usr/lib64/nagios/plugins/check_load -r -w 1.0,1.5,2.0 -c 3.0,4.0,5.0
command[check_disk]=/usr/lib64/nagios/plugins/check_disk -w 15% -c 10% /
command[check_mem]=/usr/lib64/nagios/plugins/check_mem -w 75 -c 90
command[check_total_procs]=/usr/lib64/nagios/plugins/check_procs -w 300 -c 400
command[check_swap]=/usr/lib64/nagios/plugins/check_swap -w 10 -c 5
```

Figure 31: NRPE plugins

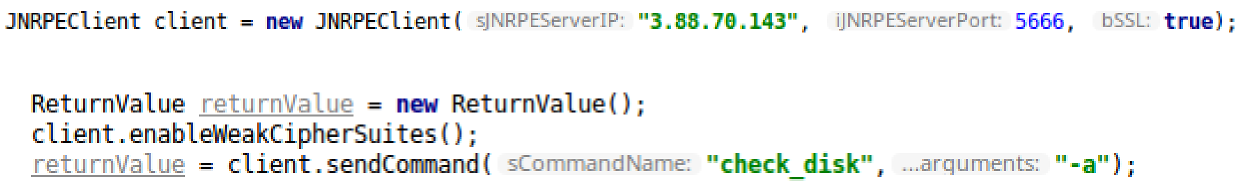
In the NRPE plugin side, we need to define the threshold values for each check as per the above image. When Java client calling it returns the value and the statue whether that value is on warning state or in a critical state.



```
<dependency>
    <groupId>net.sf.jnrpe</groupId>
    <artifactId>jcheck_nrpe</artifactId>
    <version>2.1.0-SNAPSHOT</version>
</dependency>
```

Figure 32: Java dependency for NRPE

To use the NRPE client in Java, we need to get the libraries to call the NRPE service. The above artifact facilitate the import required dependency class to the java program. As per the below, developers can initialize the JNRPEClient by giving the IP, port and other necessary arguments.

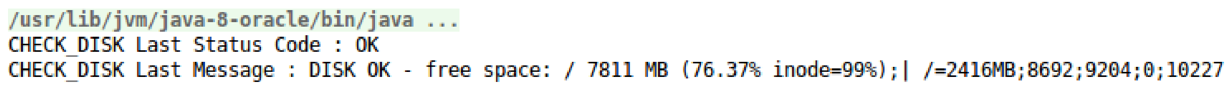


```
JNRPEClient client = new JNRPEClient( sJNRPEServerIP: "3.88.70.143",  iJNRPEServerPort: 5666,  bSSL: true);

  ReturnValue returnValue = new ReturnValue();
  client.enableWeakCipherSuites();
  returnValue = client.sendCommand( sCommandName: "check_disk", ...arguments: "-a");
```

Figure 33: Creating JNRPE client

After initializing the client, can send the comments to check the server status as per the above. We can send the commands with the argument.

- Check_disk
- Check_mem
- Check_load
- Check_users



```
/usr/lib/jvm/java-8-oracle/bin/java ...
CHECK_DISK Last Status Code : OK
CHECK_DISK Last Message : DISK OK - free space: / 7811 MB (76.37% inode=99%);| /=2416MB;8692;9204;0;10227
```

Figure 34: Results for JNRPE

After sending the commands from the java method, NRPE plugin installed in the remote server sends the server status according to the send command as per the above. Likewise, we can get the remote server status to the java client.

3.6 Architecture Design

The high overview of the proposed system is to realize cost-effective auto-scaling components between server and serverless cloud. The proposed system is monitoring all server's statistics such as RAM, CPU usage, load average and disk utilization. Based on the server stats proposed system is capable for handling traffic between servers and serverless functions. When the server resource are fully utilized, that application not be able to handle the more traffic. If they were going to handle this, it would cause system downtimes. A possible scenario is to scale the servers horizontally or vertically. If we provide the cloud servers, it's very costly. If they run on those servers in on promise, it is very difficult to scale.

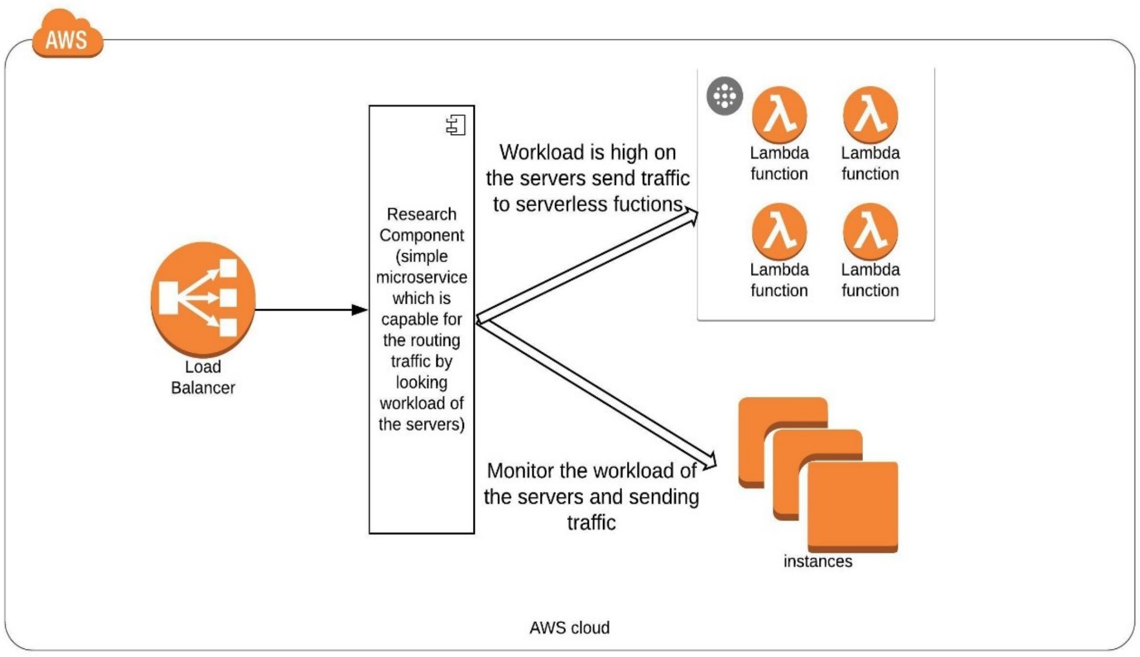
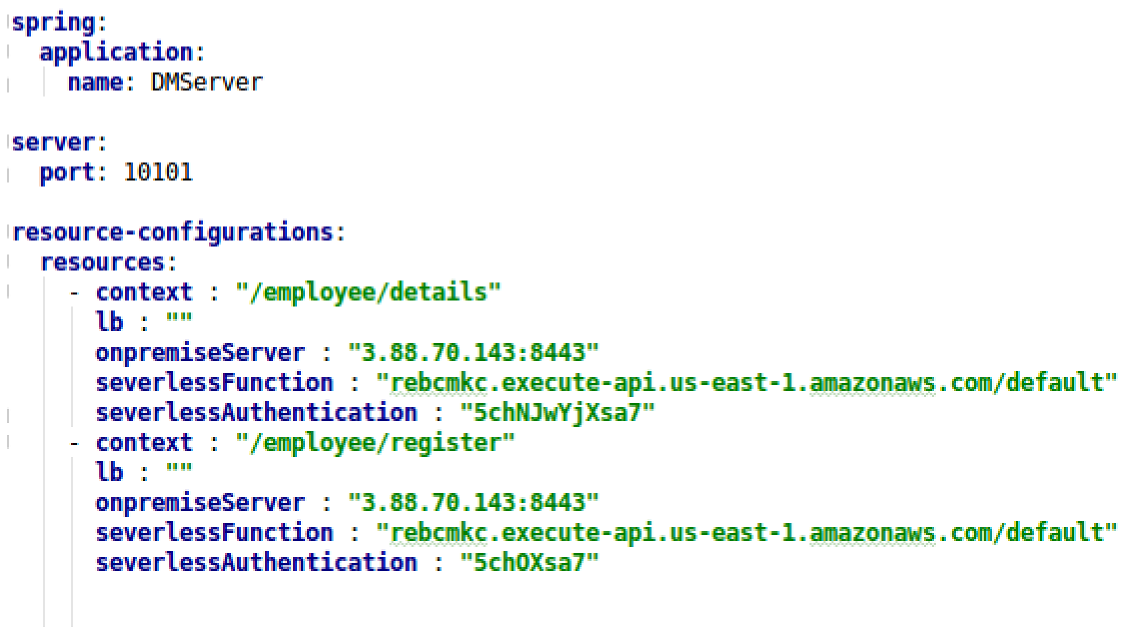

AWS
Load Balancer
Research Component (simple microservice which is capable for the routing traffic by looking workload of the servers)
Workload is high on the servers send traffic to serverless fuctions
Lambda function
Lambda function
Lambda function
Lambda function
Monitor the workload of the servers and sending traffic
instances
AWS cloud

Figure 35: High-level architecture

Research component has the ability to monitor the server stats and get the decision to route the specific traffic to serverless functions. Those decisions are not round-robin load balancing it get based on the priorities. Serverless function cost is making based on the requests execute on the function. For handing this kind of huge traffic, you did not need to provide additional resources. Its all handled by the research component.

3.7 Implementation

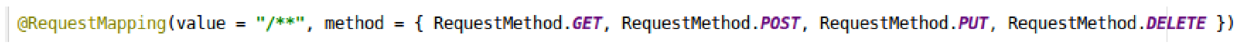
Research component is developed as a microservice using the JAVA spring boot framework. Because it should be a light weighted and should need to handle traffic than other servers handling traffic. In the spring boot microservice comes with the embedded tomcat server so that we no need additional applications to host this research component.



```yaml
spring:
  application:
    name: DMServer

server:
  port: 10101

resource-configurations:
  resources:
    - context : "/employee/details"
      lb : ""
      onpremiseServer : "3.88.70.143:8443"
      severlessFunction : "rebcmkc.execute-api.us-east-1.amazonaws.com/default"
      severlessAuthentication : "5chNJwYjXsa7"
    - context : "/employee/register"
      lb : ""
      onpremiseServer : "3.88.70.143:8443"
      severlessFunction : "rebcmkc.execute-api.us-east-1.amazonaws.com/default"
      severlessAuthentication : "5chOXsa7"
```

Figure 36: Configuration file

As per the above, each other server IP address are needs to configure in the configuration file on microservice. In the research, component reads the values form the NRPE commends which are send by other serves [30]. Those results consist of respective server statistics such as memory usage, CPU usage, load average and disk utilization [31]. Based on those values research component will choose what the works are should need to divide to the serverless functions.



```java
@RequestMapping(value = "/**", method = { RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE })
```

Figure 37: Allow all traffic

In microservices, it cater the all the traffic and map with the configuration file for resources. If API context map with the API resource, then microservices get the request inside otherwise, it returns an error.

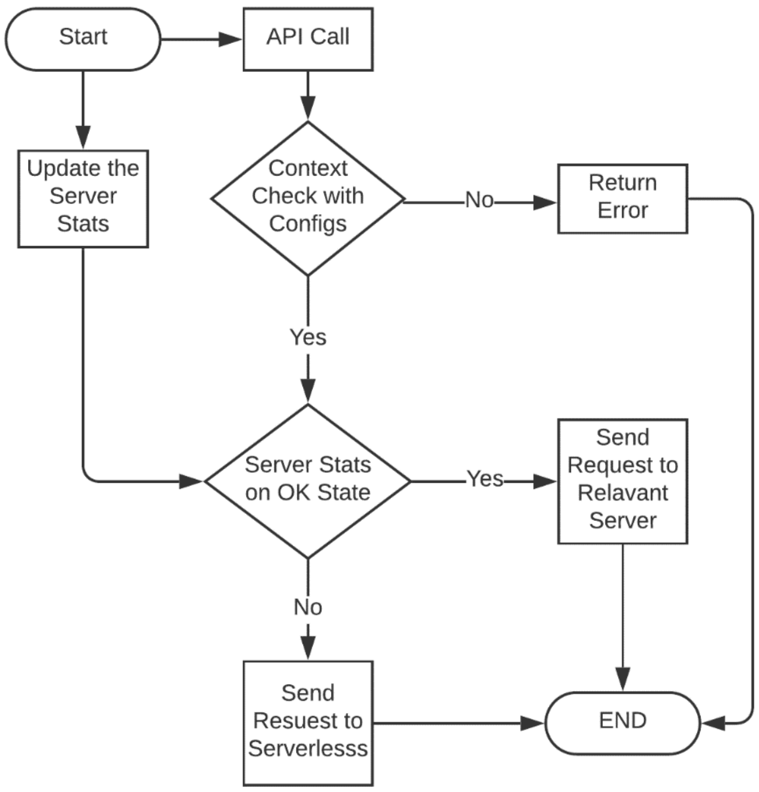
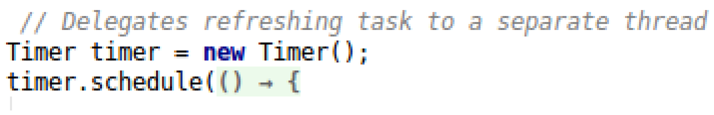


Figure 38: Logical flow diagram

Based on the server statistics research module come up with the decision on the traffic routing. If the server statistics such as memory, CPU usage and the disk usage is on the warning state, then traffic route to the serverless function. Before routing to the serverless side, the application request got intercepted to make that request according to the serverless function such as add authentication header, add content types and so on.



```
 // Delegates refreshing task to a separate thread
Timer timer = new Timer();
timer.schedule(() → {
```

Figure 39: Schedule tasks

In the application, use the separate Java time schedulers for each server to get the server details and update the global variable on configured time gaps. In this implementation its running as separate threads and worked as a non-blocking separate work for API path.

3.8 Integration to Existing Systems

The most critical path is how this research component is working with the existing systems. Value addition in this component can work with the existing system and load balance between the server and the serverless. In this research will explain the architectural-wise integration using the one use case.

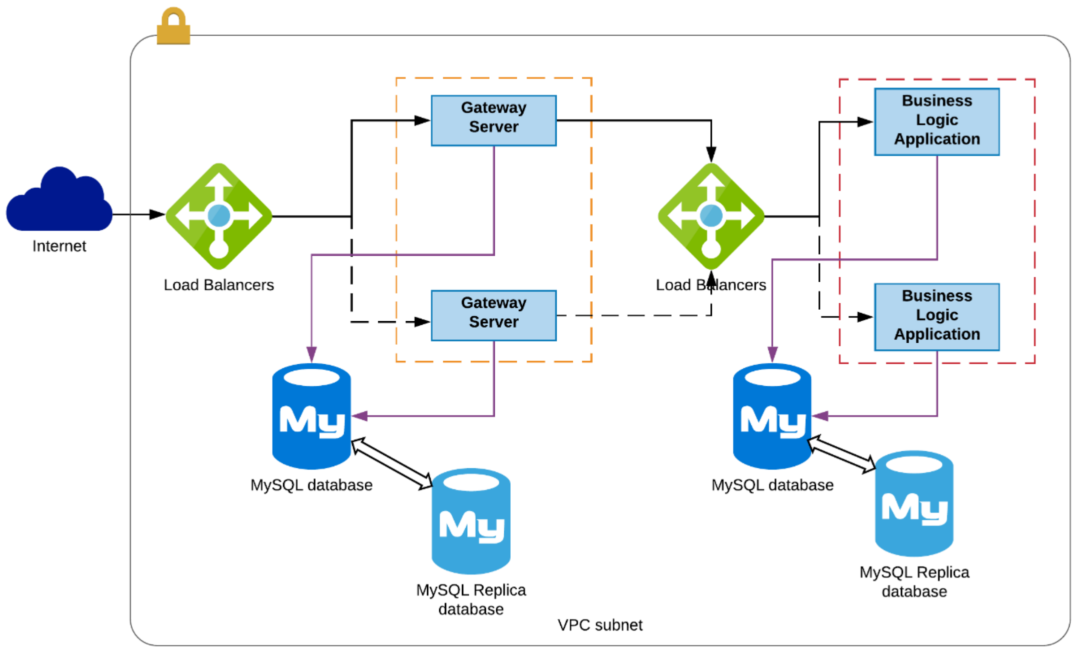


Figure 40: Existing system architecture diagram

In the above architecture (figure 40) deployed on the on-premise servers running with the monolithic application. In that front end load balancer only have access with the internet and it placed on the demilitarized zone. Other all the VMs are deployed on the non - demilitarized zone which don't have any incoming traffic from the external parties. This architecture is fully running on the active-passive mode; hence it brings the HA to the whole system. In this setup DB also configured as a master-slave so DB layer also having high availability. Used the internal load balancer to the load balancer the traffic between the business logic. Those all VMs are deployed on the same subnet network so that can reduce the network latency also. In this scenario, business owners need to manage the lot of VM and need to run the manage services for the whole setup and that will cost a lot.

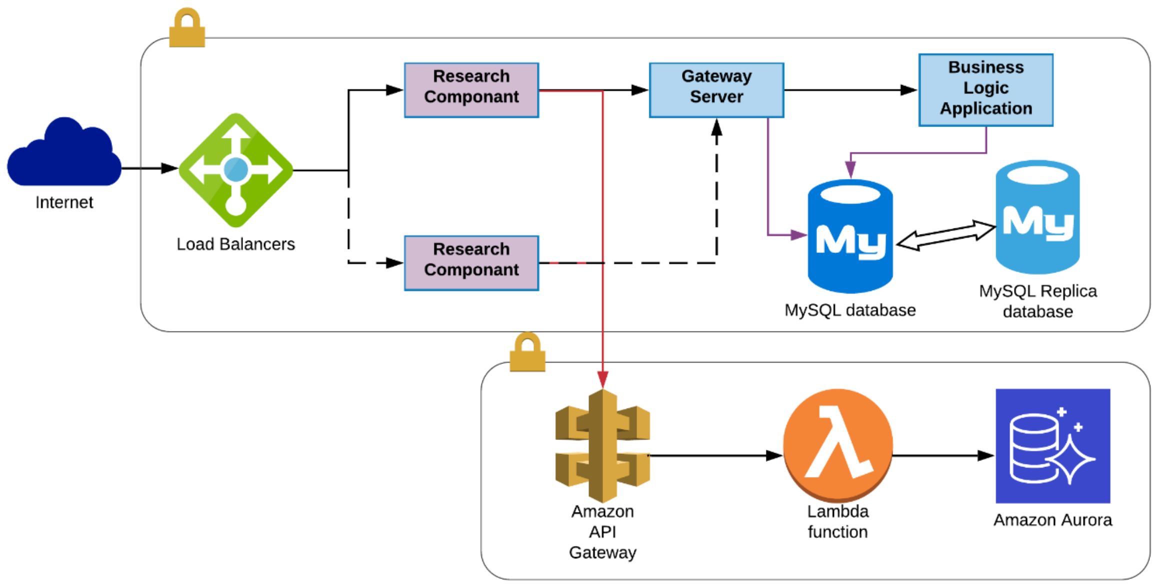


Figure 41: Architecture with new component

To overcome the limitation and seamless migration, the service to the serverless will help this architecture. In the above figure show how the research component plugs into the existing system. Using the API calls from the external load balancer research component, check the traffic with the provisioned or not. If it is a valid traffic research component will route traffic to the serverless function or on-premise servers. All the high availability and the manager service are managed by the cloud provider and the cloud database also maintained by the cloud provider.

CHAPTER 4
RESULT AND EVALUATION

4.1 Introduction

Before this chapter, we discussed how the research component behaves on the real environment, and it will help to understand how each section on the research component works according to the archive the outcome. All system need to be tested and evaluated with respect to its non-functional requirements and functional requirements. That will help to produce the bug-free software to the as a final output. System evaluation help to set the system benchmarks as well as the achieve the goal into the expected level. In this chapter, discuss how system tested and how the system evaluates on the real environment to capture the actual goal.

4.2 Testing on System

Testing is a necessary process to ensure that system function is up to the standard level on the acceptance criteria. The proper testing process brings the procedure to check the functional and the non-function requirements in the system with the perspective of both developer and the end-user. That means the system needs to cater to the end-user goal by with proper software development process. A small mistake on the development process can cause for whole system crash, and maybe that will not reflect the final system and the end-user. Do the testing on each unit on the development time also an essential task for the researcher. After the system developed need to run the test to make sure all system work as expected as well as the connected systems also working as expected with the new research component.

Performance: How system response to the input. It depend on the external factors.

Accuracy: Software must be ensured to give full functionality correctly.

Functionality: Proper software should be able to achieve the required functional requirements. Test the system functional requirement is the most crucial task on the project. Uncertainty, the developed software is not capable of achieving its functional requirements; it is useless.

4.3 Test Strategies and Procedure

The previous topic discusses the importance of the testing of the newly developed software. To capture the software, each drawback testing needs to done as a sequential manner. In the first stage, testing need to start with the unit testing on each software unit. After unity testing done without any defects, then the tester can move to the second phase, which is integration testing. In the second phase, the tester needs to integrate the subsystem to the developed system and need to start the whole end to end testing. In this phase need to test the software clustered setup as well as the standalone setup. Using this testing, we can identify the system defects and user acceptance criteria. After testing done with the zero defects, then can conduct the user acceptance testing with the involvement of the end-user / customer. Those testing need to cover all of the functional and the non-functional requirements on the whole system, which need to cover all components and integrated subsystems. If those test cases did not cover the whole system that ends up with the bugs in the actual production system which can disappoint the end-user/custom which used the system but didn't found the expected behaviors on the system. To get the bug free research component researcher to conduct the testing in a conventional manner. The rest of this chapter will discuss that.

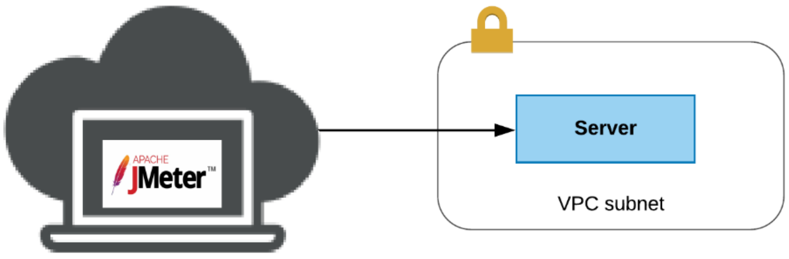


Figure 42: Load test with Jmeter

Table 3: Server specification

Server Type	**Purpose**	**RAM**	**CPU**	**OS**
HP Laptop	Testing Client (Jmeter)	16	4	Linux - Mint
t2.micro (AWS)	Business Logic Hosted	1	1	RH
t2.medium (AWS)	Business Logic Hosted	4	2	RH

Using the above resources and the architecture did an integration testing with the servers. For that used the Jmeter client for generating the traffic with the different TPS.

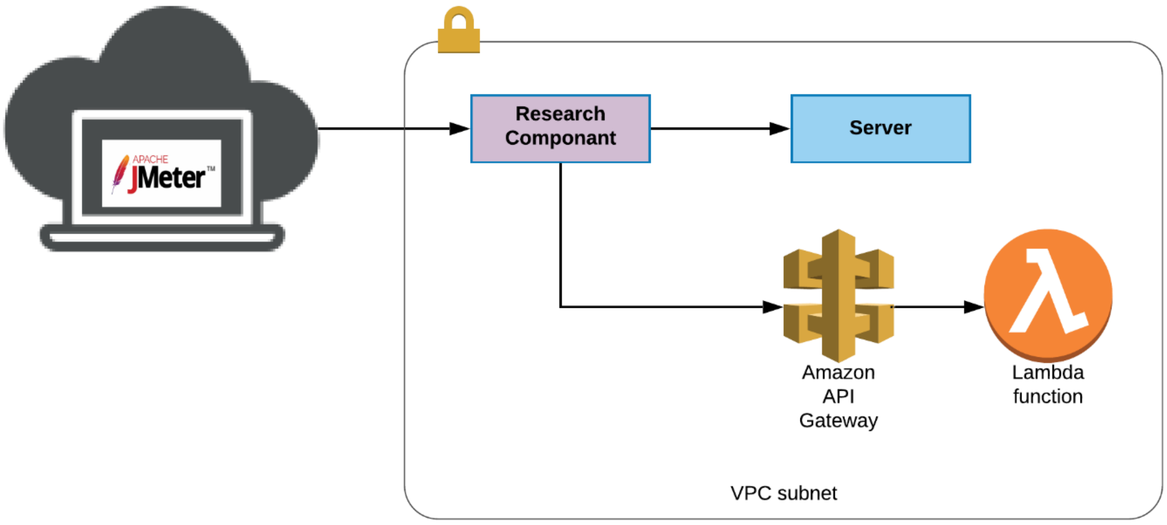


Figure 43: Testing with research component

Table 4: Server specification (with research component)

Server Type	**Purpose**	**RAM**	**CPU**	**OS**
HP Laptop	Testing Client (Jmeter)	16	4	Linux - Mint
t2.micro (AWS)	Research Component	1	1	RH
t2.micro (AWS)	Business Logic Hosted	1	1	RH
t2.medium (AWS)	Business Logic Hosted	4	2	RH

Changing the service type and did the load test with the different TPS and collect the statistics for evaluation purpose.

4.4 Evaluation

Research component deployed in the real environment and evaluated on two main criteria.

- Performance
- Cost

During the testing collect the CPU usage, memory usage and the disk usage statistic under the different TPS and different server specs. To evaluate the performance of the research component did testing with the without research component and took the above statistics to do the comparison.

4.4.1 Performance Evaluation

Test case 1 – Without research component

In this test used the business logic without the research component and collected the statistic with 20TPS traffic generated from the jmeter which is place on the remotely.

Table 5: Test case 1 details

Instance Type	**TPS**	**Duration**	**CPU**	**RAM**
t2.micro	20	30min	1	1GB

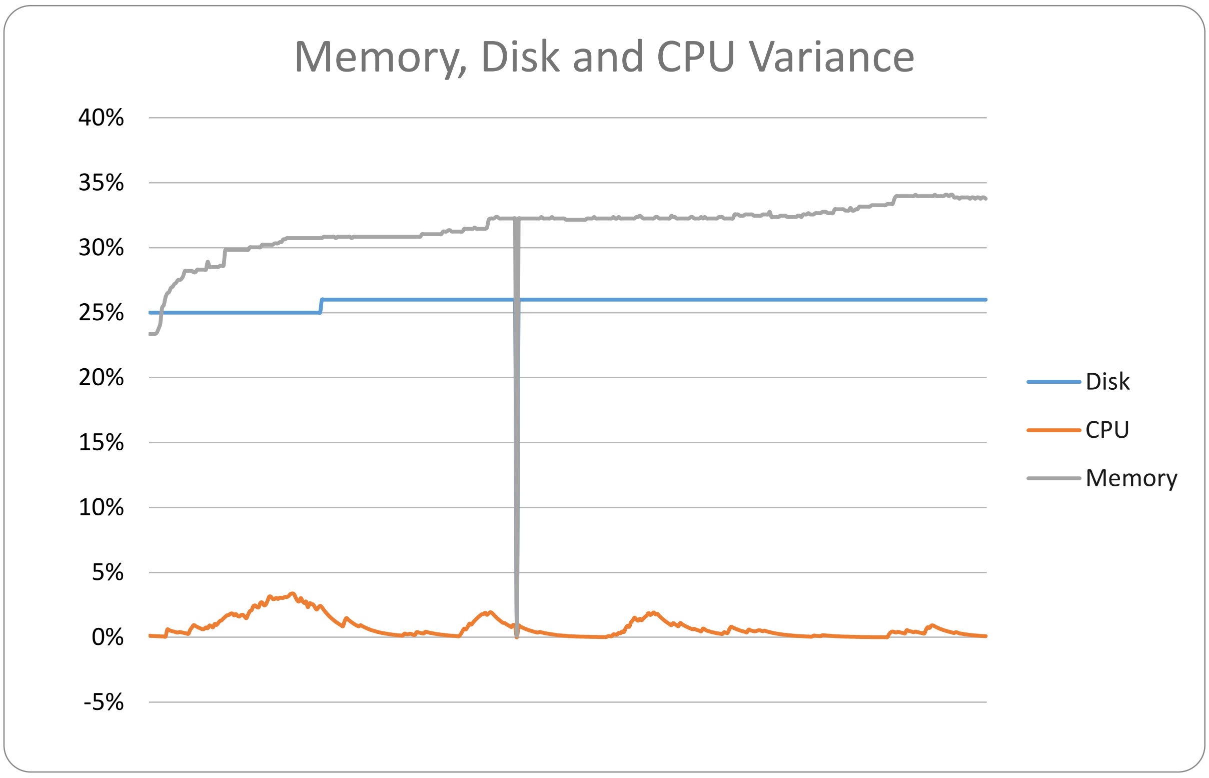


Figure 44: Graph of statistics for test case 1

Test case 1 – With research component

In this test used the business logic and the research component in a hybrid way and collected the statistic with 20TPS traffic generated from the jmeter which is placed remotely.

Table 6: Test case 1 with research component details

Instance Type	**TPS**	**Duration**	**CPU**	**RAM**
t2.micro	20	30min	1	1GB
With AWS API gateway and lambda function				

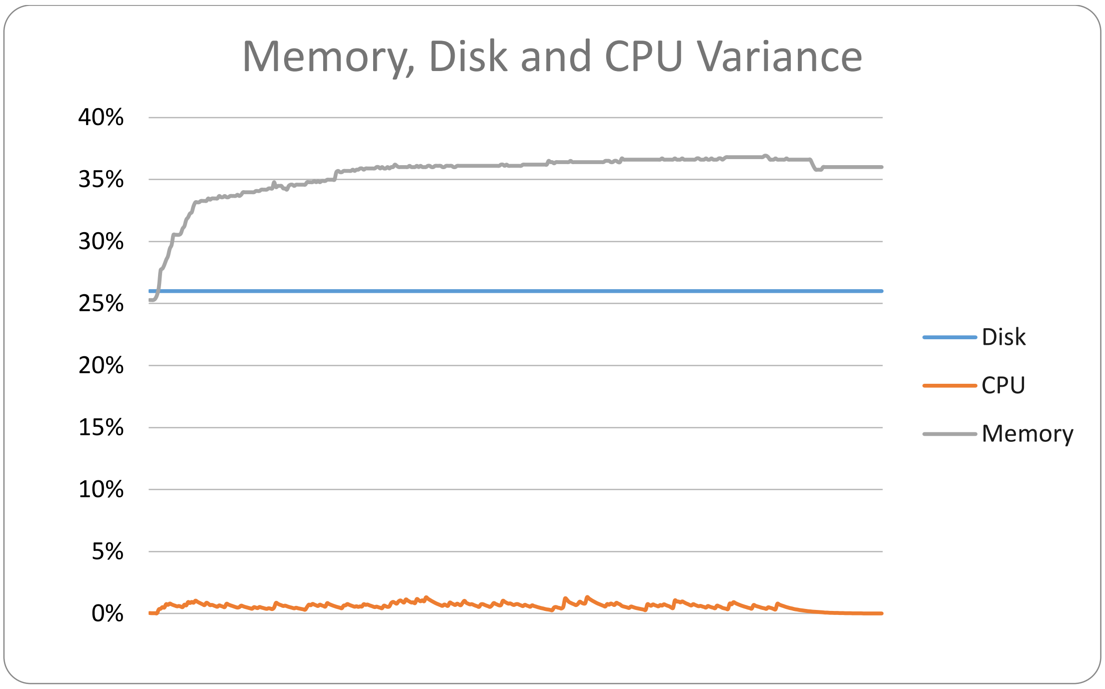


Figure 45: Graph of statistics for test case 1 with research component

Table 7: Request count on test case 1

Server Request Count	32658
Serverless Request Count	1918

Test case 1 – Summery

By comparing the above two graphs with the research component graph is pretty good than the other. In the first graph can see the memory drop and that time traffic got interrupted and also CPU is got very high during the load testing.

Test case 2 – Without research component

In this test used the business logic without the research component and collected the statistic with 50TPS traffic generated from the jmeter, which is placed on the remotely.

Table 8: Test case 2 details

Instance Type	TPS	Duration	CPU	RAM
t2.micro	50	30min	1	1GB

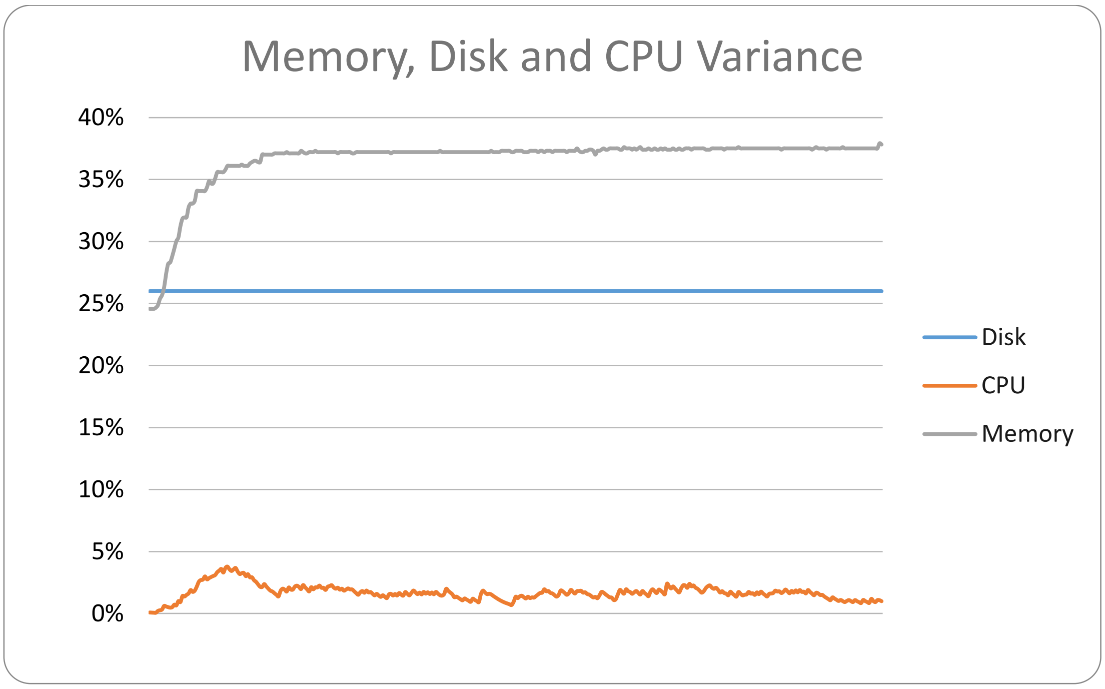


Figure 46: Graph of statistics for test case 2

Test case 2 – With research component

In this test used the business logic and the research component in a hybrid way and collected the statistic with 50TPS traffic generated from the jmeter, which is placed on the remotely.

Table 9: Test case 2 with research component details

Instance Type	TPS	Duration	CPU	RAM
t2.micro	50	30min	1	1GB
With AWS API gateway and lambda function				

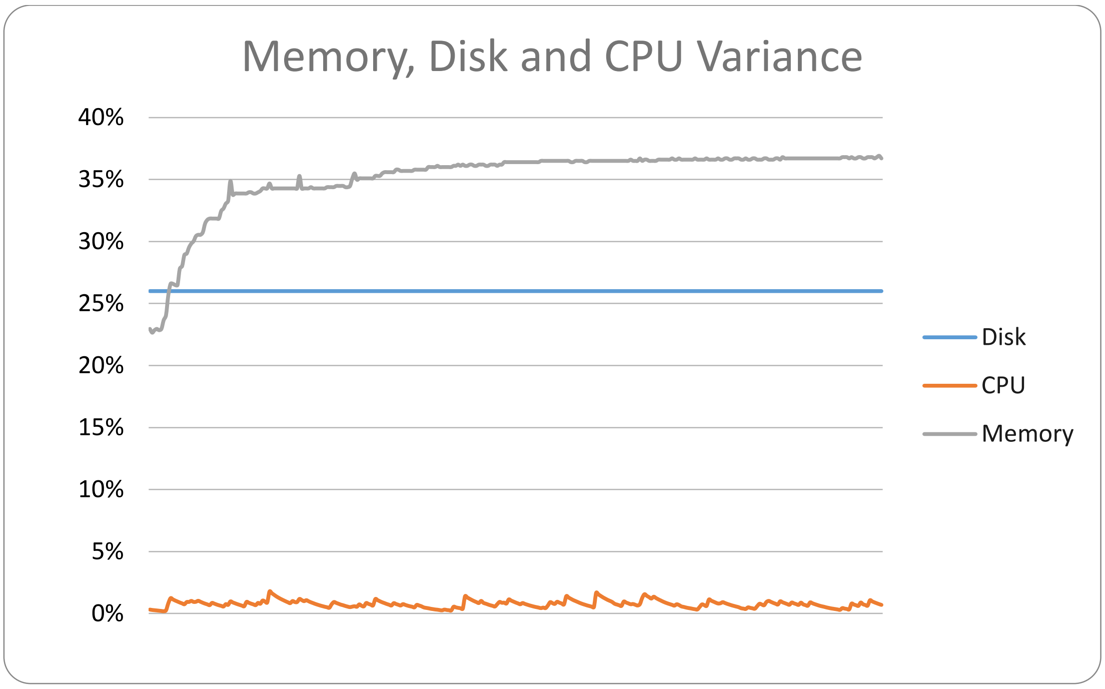


Figure 47: Graph of statistics for test case 2 with research component

Table 10: Test case 2 request count

Server Request Count	85861
Serverless Request Count	4863

Test case 2 – Summary

By comparing the above two graphs with the research component graph is pretty good than the other. In the first graph, we can see the high CPU and memory. But when come to the solution with the research component CPU and the memory is stable.

Test case 3 – Without research component

In this test used the business logic without the research component and collected the statistic with 100TPS traffic generated from the jmeter which is placed remotely.

Table 11: Test case 3 details

Instance Type	TPS	Duration	CPU	RAM
t2.micro	100	30min	1	1GB

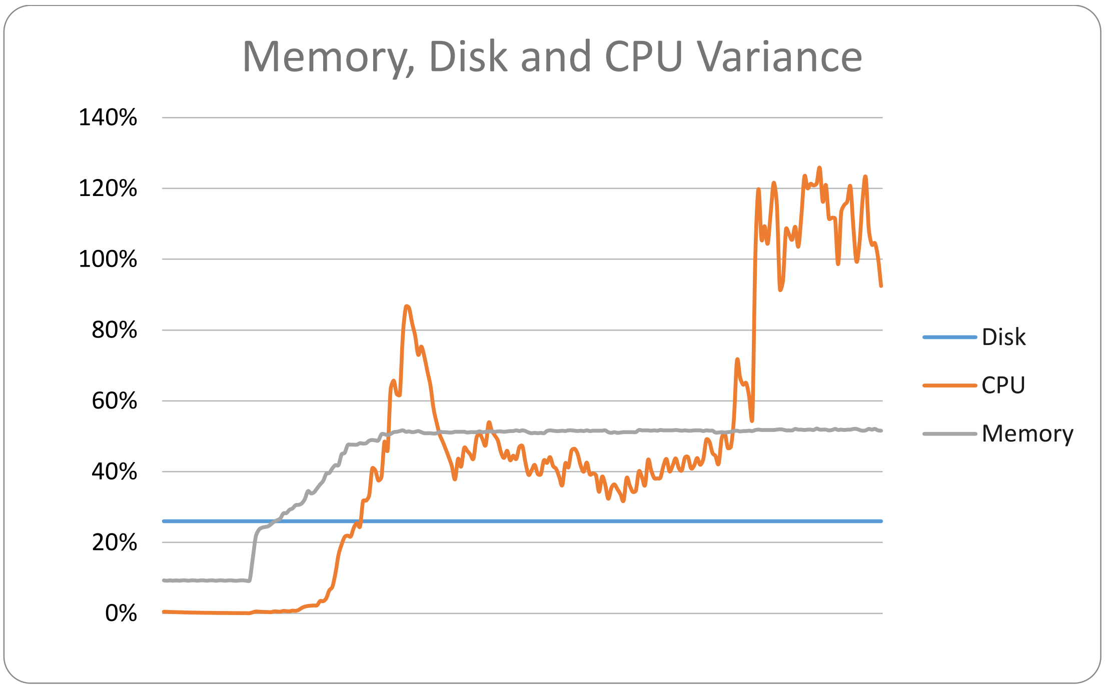


Figure 48: Graph of statistics for test case 3

Test case 3– With Research Component

In this test used the business logic and the research component in a hybrid way and collect the statistic with 100TPS traffic generated from the jmeter which is placed remotely.

Table 12: Test case 3 with research component details

Instance Type	TPS	Duration	CPU	RAM
t2.micro	100	30min	1	1GB
With AWS API gateway and lambda function				

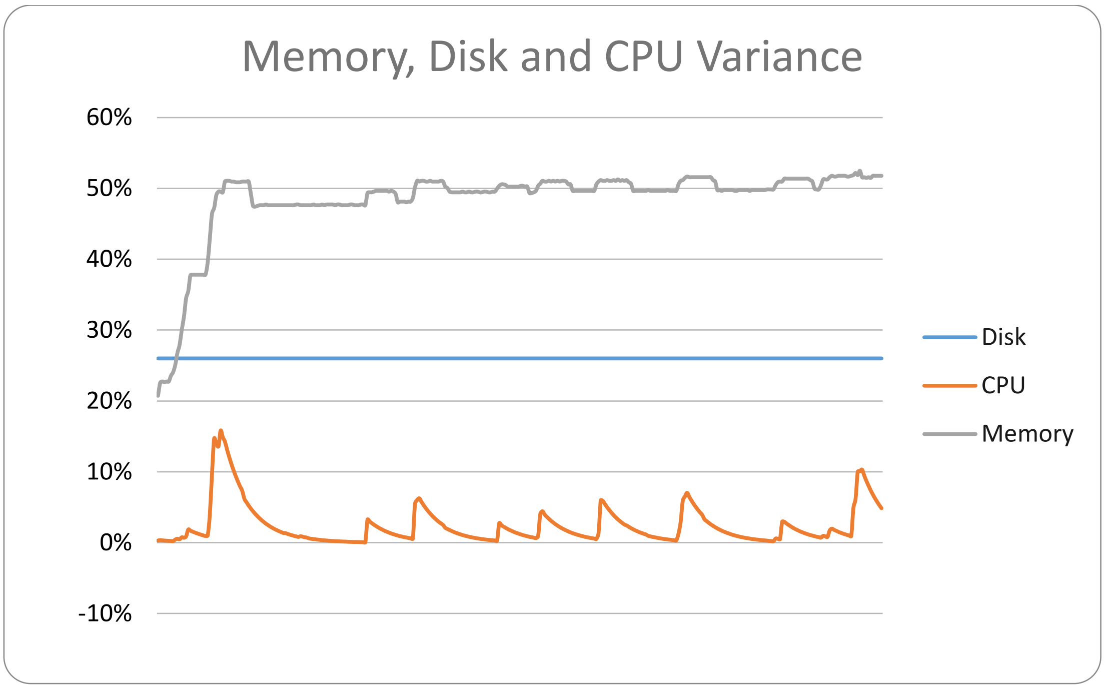


Figure 49: Graph of statistics for test case 3 with research component

Table 13: Test case 3 request count

Server Request Count	17653
Serverless Request Count	141858

Test case 3 – Summary

By looking at the graph, you can see the huge difference on the CPU load on between without and with the research component. Research component help reduce the server load by load balance between server and serverless functions.

Test case 4 – Without research component

In this test used the business logic without the research component and collected the statistic with 100TPS traffic generated from the jmeter, which is placed on the remotely.

Table 14: Test case 4 details

Instance Type	TPS	Duration	CPU	RAM
t2.medium	100	30min	2	4GB

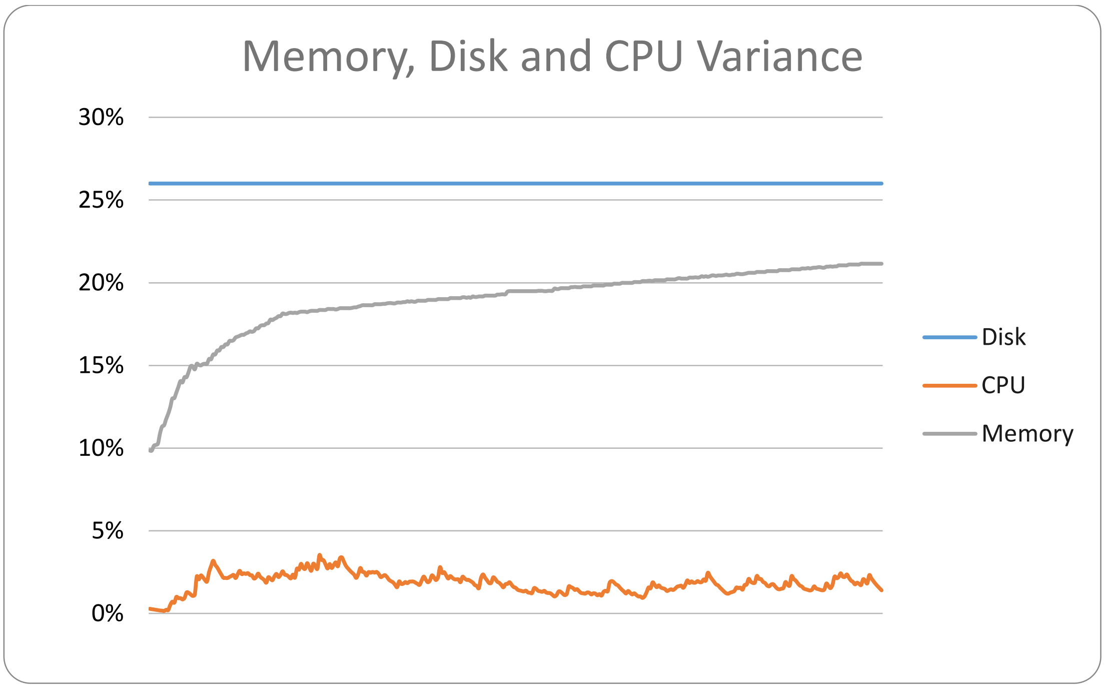


Figure 50: Graph of statistics for test case 4

Test case 4 – With Research Component

In this test used the business logic and the research component in a hybrid way and collected the statistic with 100TPS traffic generated from the jmeter, which is place on the remotely.

Table 15: Test case 4 with research component details

Instance Type	TPS	Duration	CPU	RAM
t2.medium	100	30min	2	4GB
With AWS API gateway and lambda function				

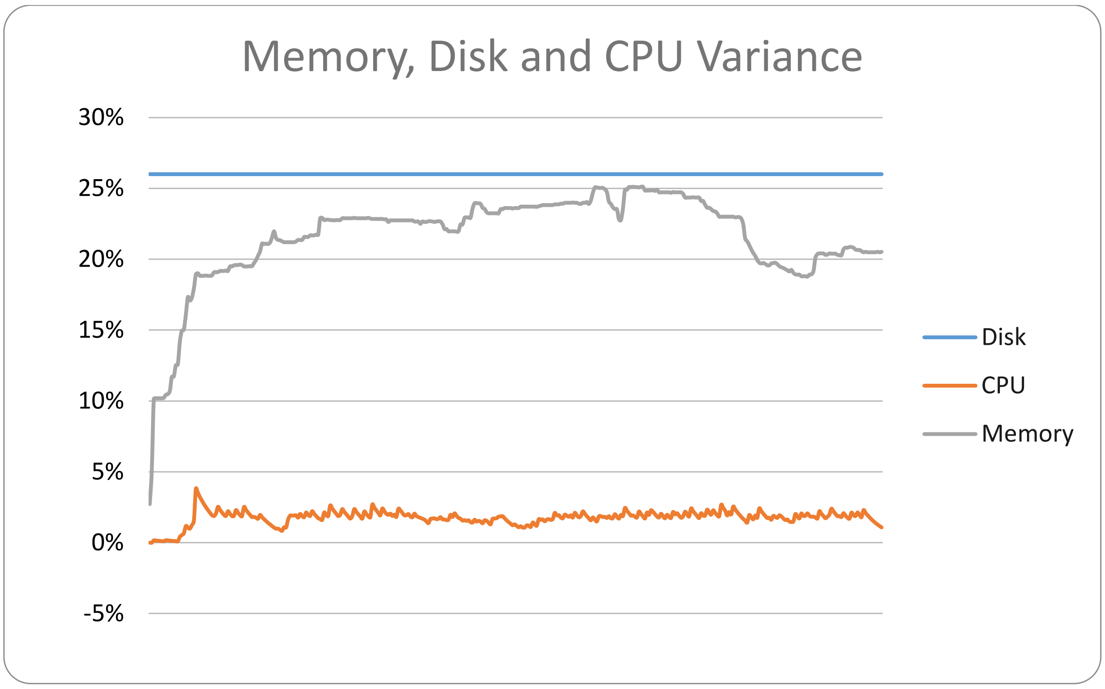


Figure 51: Graph of statistics for test case 4 with research component

Table 16: Test case 4 request count

Server Request Count	128589
Serverless Request Count	49977

Test case 4 – Summary

In this test case, we increased the server specification as follows and didn't change the TPS. We increase the RAM by 3GB and CPU by one more additional core. Then we collect the statistics with a research component and without it. By comparing those graphs with the previous test (test3) after increasing the server resources can handle the requests without any major interruptions, but sometimes CPU status got warning state. With the research component, CPU usage is quite stable by load balancing between server and serverless.

4.4.2 Cost Evaluation

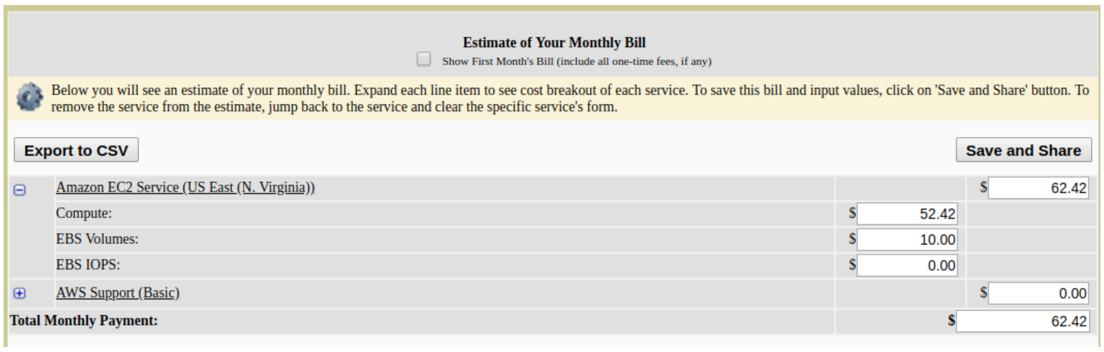
Cost is a very important factor in the business world. Because most of the organizations not moving to the new technologies because of the cost and the operational risk. We evaluate the cost for infra and the operational with the monolithic system and the research component.

Table 17: AWS cloud cost estimation

Instance Type	TPS	per 1 Month Cost		per 1 Year Cost	
		Server	Serverless Solution	Server	Serverless Solution
t2.micro	20	62.42	46.1	749.04	553.2
t2.medium	100	87.67	260.97	1,052.04	3131.64

Serverless solution cost is basically static cost as per the request count. It will not cost any other additional things. However, the people who are using the on-premise or cloud servers they will have some more cost.

1. Infrastructure maintained cost
2. Security maintain cost
3. 24*7 Support team cost for service monitoring



Estimate of Your Monthly Bill

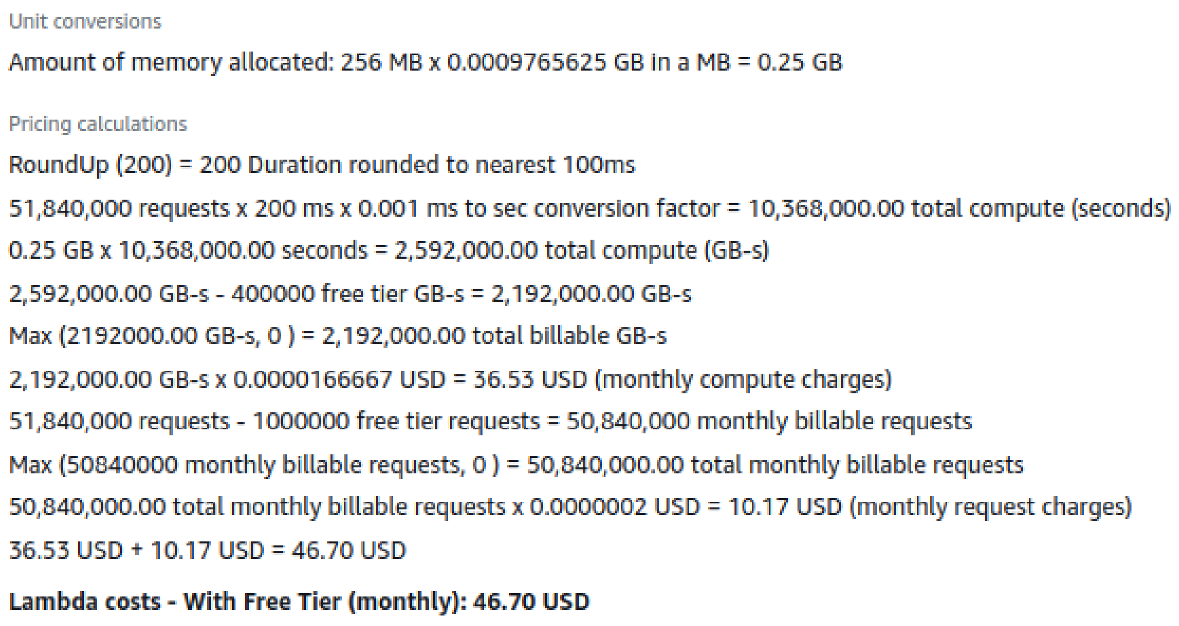
☐ Show First Month's Bill (include all one-time fees, if any)

Below you will see an estimate of your monthly bill. Expand each line item to see cost breakout of each service. To save this bill and input values, click on 'Save and Share' button. To remove the service from the estimate, jump back to the service and clear the specific service's form.

Amazon EC2 Service (US East (N. Virginia))		$ 62.42
Compute:	$ 52.42	
EBS Volumes:	$ 10.00	
EBS IOPS:	$ 0.00	
AWS Support (Basic)		$ 0.00
Total Monthly Payment:	$ 62.42	

Figure 52: Cost calculation for 20TPS

Image 52 shows the how cost involve for the 20TPS on the AWS account. Cloud provider sends the details bill to the cloud consumer then that person can evaluate it can do the necessary things to reduce the costs.



Unit conversions

Amount of memory allocated: 256 MB x 0.0009765625 GB in a MB = 0.25 GB

Pricing calculations

RoundUp (200) = 200 Duration rounded to nearest 100ms

51,840,000 requests x 200 ms x 0.001 ms to sec conversion factor = 10,368,000.00 total compute (seconds)

0.25 GB x 10,368,000.00 seconds = 2,592,000.00 total compute (GB-s)

2,592,000.00 GB-s - 400000 free tier GB-s = 2,192,000.00 GB-s

Max (2192000.00 GB-s, 0) = 2,192,000.00 total billable GB-s

2,192,000.00 GB-s x 0.0000166667 USD = 36.53 USD (monthly compute charges)

51,840,000 requests - 1000000 free tier requests = 50,840,000 monthly billable requests

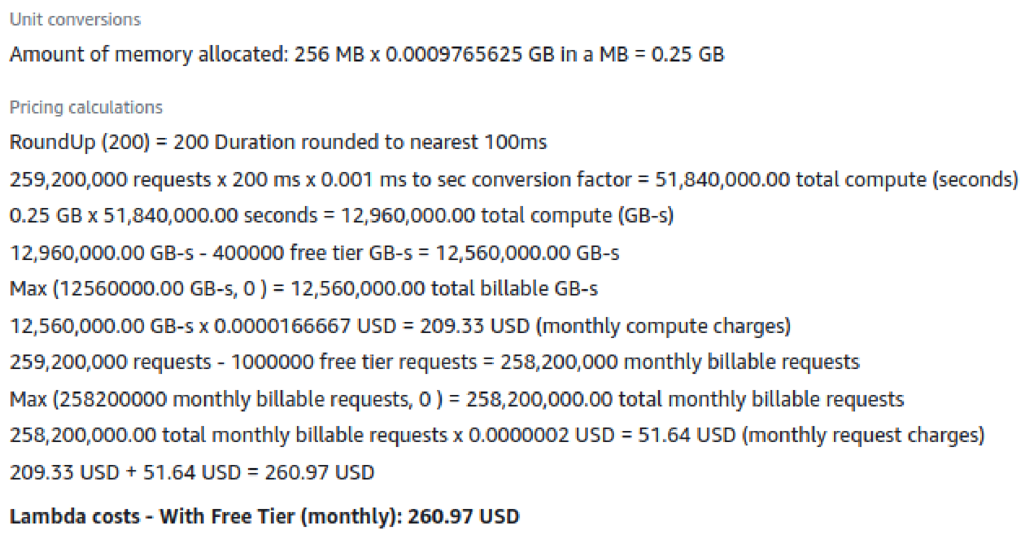
Max (50840000 monthly billable requests, 0) = 50,840,000.00 total monthly billable requests

50,840,000.00 total monthly billable requests x 0.0000002 USD = 10.17 USD (monthly request charges)

36.53 USD + 10.17 USD = 46.70 USD

Lambda costs - With Free Tier (monthly): 46.70 USD

Figure 53: Cost calculation on 20TPS in serverless



Unit conversions

Amount of memory allocated: 256 MB x 0.0009765625 GB in a MB = 0.25 GB

Pricing calculations

RoundUp (200) = 200 Duration rounded to nearest 100ms

259,200,000 requests x 200 ms x 0.001 ms to sec conversion factor = 51,840,000.00 total compute (seconds)

0.25 GB x 51,840,000.00 seconds = 12,960,000.00 total compute (GB-s)

12,960,000.00 GB-s - 400000 free tier GB-s = 12,560,000.00 GB-s

Max (12560000.00 GB-s, 0) = 12,560,000.00 total billable GB-s

12,560,000.00 GB-s x 0.0000166667 USD = 209.33 USD (monthly compute charges)

259,200,000 requests - 1000000 free tier requests = 258,200,000 monthly billable requests

Max (258200000 monthly billable requests, 0) = 258,200,000.00 total monthly billable requests

258,200,000.00 total monthly billable requests x 0.0000002 USD = 51.64 USD (monthly request charges)

209.33 USD + 51.64 USD = 260.97 USD

Lambda costs - With Free Tier (monthly): 260.97 USD

Figure 54: Cost calculation on 100TPS in serverless

Above two images (figure 53 & figure 54) show the how AWS cost for the serverless in a detail manner. According to that we can analyze the cost and can abject the code line according to reduce the cost.

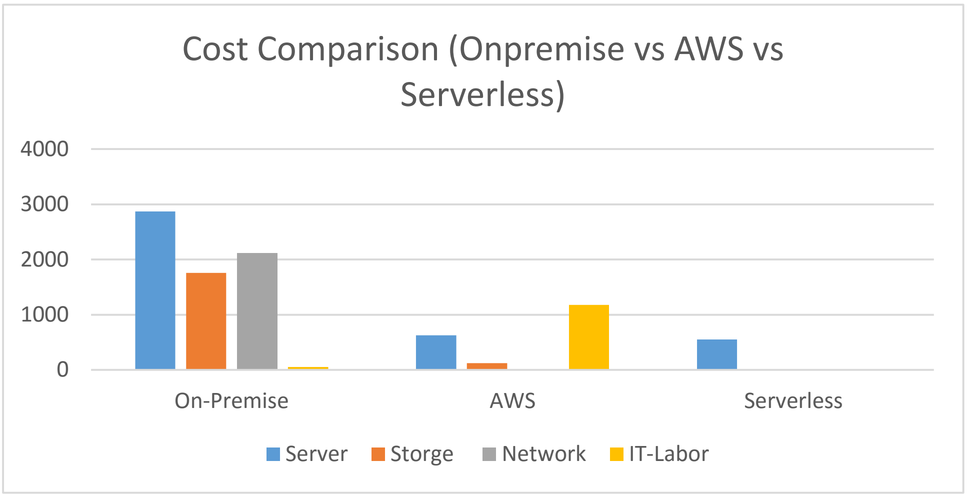


Figure 55: Graph on cost comparison

By looking at the cost on the On-premise, cloud and the serverless we can conclude that cost-effective method is serverless solution. But the problem organizations did not move from the monolithic to the serverless. So by combining the serverless and the servers are the good initiative to move older technology legacy system to the new technologies.

CHAPTER 4
CONCLUSION

4.1 Introduction

In the previous chapters, discuss other researches works and as well the developed research component in an in-depth. This chapter brings the facts to evaluate the research hypothesis, which is load balancing between the server and the serverless is a cost-effective way. For that research conduct, the testing can collect the data to evaluate the system up to the standard way. Those details will elaborate on upcoming topics with some limitations and future works.

4.2 Problems Faced and the Benefits of the Research Module

These days, some of the companies using the monolithic application for their businesses. Nevertheless, most of the companies moving from the monolithic application to the microservice bases applications or serverless architecture. People need new technologies to get more revenue from the business. New technologies cannot be embedded into legacy systems because those are not supported, and those systems are tightly coupled. People are afraid that moving the whole system to the new platform directly. The people who are using the legacy systems, they always used the horizontally and vertically scaling to absorb more traffic. If their system on the cloud environment, it is very costly for them. However, if their system deployed in the on-premise server, it is very difficult to increase the server instance and increase the server resources (number of cores, RAM). Hence there are facing much trouble because of that. In this research proposed a system that capable of switching traffic between serverless and the servers. It's something like cost-effectively scaling the system and not need to vertically and horizontally scale the system. There are several advantages of using this research component to the business owners. Then organization who are willing to go for the serverless functions they can use this research component and can run their system in a hybrid way. Another advantage can cost-effectively scale the system when traffic spicks according to the server loads.

4.3 Revisited the Objectives

In the final stage in the last chapter on this thesis will discuss the objectives and its success criteria in briefly. Using this can come to the conclusion about the research.

Objective 1: Analyze the current cloud provider's services.
Nowadays, there are having a large number of cloud providers in the technological world. Such as Google, AWS, Alibaba, Azure and etc. In this research does the literature review related to the existing cloud providers and what are the services they have provided to the cloud consumers.

Objective 2: Deeply investigate the cost of maintaining both on-premise servers and cloud servers.
This thesis clear showed how the cost calculated on the servers and the serverless by providing the calculations for every single level. In the on-premise deployment cost factors are deeply reviewed and illustrated in the previous chapter.

Objective 3: Prediction of server workloads based on current server statistics.
Based on the current workloads on the servers, the research component able to load balance traffic. Because of the server resources on the warning state, it can become the critical state in a short time of period. Hence on the warning state traffic routed to the serverless to prevent the traffic failures.

Objective 4: Analyze server resource statistics with respect to their capabilities (Load average, memory, and disk) and cost.
Using the NREP technology, get the remote server details to the research component and doing the analysis. The research component used the JNRPE java libraries to get the server stats. The previous chapter shows that the cost analysis on this solution is good than maintaining the monolithic system.

Objective 5: Developing a component to load balance the traffic base of the current server workload.
Developed a component using the Java Spring boot framework to load balance the traffic based on the remove server loads. The methodology chapter described the implementation of this research component.

Objective 6: Analyze the integration points to integrate newly developed component with the monolithic systems.
The most important part is how research components plug into the existing systems. Providing the actual use cases shows how the research component integrates into the existing systems.

Objective 7: Deeply go through the performance on the microservices.
Deeply investigate the performance of the microservices and its features. After that, choose the Java Spring boot framework, which is used for developing the enterprise software in the software industry.

Objective 8: Analyze a method for deploying existing microservices on the serverless.
Under this objective by wrapping the existing microservices with the hander and adding some dependencies can easily deploy the existing microservices on the serverless platform. This thesis clearly showed how to do that in step by step on previous chapters.
By looking at all objectives, this research satisfies all the mention objectives in detail manner. Therefore, the author has proven the research hypothesis.

4.4 Limitations

Currently, the main limitation is for adding new API to the research component that service need to restart. So in that short time period traffic can be lost if they don’t have the active passive deployment.

4.5 Further works

In this thesis presents the ongoing technology research to the cloud and load balancing. Those fields are vastly evolving areas and this research can continue with the following areas,

- Adding a machine-learning algorithm to predict future loads
- Improve the load balancing by considering error descriptions, response times

4.6 Summary

The major objectives of this research and the limitations of this research component mention in this section. To continue this research this section gives further works as a foundation to another research. Using that anyone can enhance this system. Using the dynamic scheduling methodology, build the research component to balance the load between server and serverless cloud with cost-effective way. Hence the aim of this research is satisfied. Using this approach showed that can scale the existing system to cater the sudden traffic loads on low-cost manner and without having any downtimes or traffic interruptions with giving the analytics results. Using this hybrid strategy, business owners can evaluate their monolithic system on the new serverless technologies. In this research critically evaluated the research component and make sure this research accomplish the research objectives and the aim with respect to the research hypothesis.

REFERENCES

[1] R. F. El-Gazzar, “A Literature Review on Cloud Computing Adoption Issues in Enterprises,” in *Creating Value for All Through IT*, vol. 429, B. Bergvall-Kåreborn and P. A. Nielsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 214–242.

[2] J. S. Hurwitz, R. Bloor, M. Kaufman, and F. Halper, *Cloud Computing For Dummies*. John Wiley & Sons, 2010.

[3] “Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds - ScienceDirect.” [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X12001008

[4] T. P. Dufficy, “What is Private Cloud? Advantages and Disadvantages.” [Online]. Available: http://www.serverspace.co.uk/blog/what-is-private-cloud-plus-advantages-disadvantages.

[5] “What is hybrid cloud? - Definition from WhatIs.com,” *SearchCloudComputing*. [Online]. Available: https://searchcloudcomputing.techtarget.com/definition/hybrid-cloud.

[6] I. Baldini *et al.*, “Serverless Computing: Current Trends and Open Problems,” *ArXiv170603178 Cs*, Jun. 2017.

[7] “AWS Lambda – Serverless Compute - Amazon Web Services,” *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/lambda/.

[8] “Serverless Computing,” *Google Cloud*. [Online]. Available: https://cloud.google.com/serverless/

[9] “Build the future of Open Infrastructure.,” *OpenStack*. [Online]. Available: https://www.openstack.org/.

[10] A. Beloglazov, S. F. Piraghaj, M. Alrokayan, and R. Buyya, “Deploying OpenStack on CentOS Using the KVM Hypervisor and GlusterFS Distributed File System,” p. 47.

[11] R. Kumar, “OpenStack Juno Release Includes Features of NFV, Big Data,” no. 2, p. 4, 2014.

[12] M. Mahjoub, A. Mdhaffar, R. B. Halima, and M. Jmaiel, “A Comparative Study of the Current Cloud Computing Technologies and Offers,” in *2011 First International Symposium on Network Cloud Computing and Applications*, Toulouse, France, 2011, pp. 131–134, doi: 10.1109/NCCA.2011.28.

[13] E. Caron, L. Toch, and J. Rouzaud-Cornabas, “Comparison on OpenStack and OpenNebula performance to improve multi-Cloud architecture on cosmological simulation use case,” p. 24.

[14] A. Mehta and D. S. N. Panda, “Design of Infrastructure as a Service (IAAS) Framework with Report Generation Mechanism,” vol. 13, no. 2, p. 5, 2018.

[15] “Azure Functions—Serverless Architecture | Microsoft Azure.” [Online]. Available: https://azure.microsoft.com/en-us/services/functions/.

[16] “Cloud Functions - Event-driven Serverless Computing | Cloud Functions,” *Google Cloud*. [Online]. Available: https://cloud.google.com/functions/.

[17] G. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, *Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research*. 2017.

[18] Hyungro Lee, K. Satyam, and G. C. Fox, “Evaluation of Production Serverless Computing Environments,” 2018, doi: 10.13140/rg.2.2.28642.84165.

[19] U. Rencuzogullari and S. Dwarkadas, “A technique for adaptation to available resources on clusters independent of synchronization methods used,” in *Proceedings International Conference on Parallel Processing*, 2002, pp. 385–394, doi: 10.1109/ICPP.2002.1040895.

[20] D. L. Eager, E. D. Lazowska, and J. Zahorjan, “Adaptive load sharing in homogeneous distributed systems,” *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 5, pp. 662–675, May 1986, doi: 10.1109/TSE.1986.6312961.

[21] J. F. Garamendi and J. L. Bosque, “Parallel Implementation of Evolutionary Strategies on Heterogeneous Clusters with Load Balancing,” in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, Washington, DC, USA, 2006, pp. 242–242.

[22] R. K. Yadav, A. K. Mishra, P. Navin, and S. Himanshu, “An Improved Round Robin Scheduling Algorithm for CPU scheduling,” *ResearchGate*, vol. 2, no. 4, Jul. 2010.

[23] H. Rahmawan and Y. S. Gondokaryono, “The simulation of static load balancing algorithms,” in *ResearchGate*, 2009, vol. 2, pp. 640–645, doi: 10.1109/ICEEI.2009.5254739.

[24] G. Kanagaraj, N. Shanmugasundaram, and S. Prakash, “Adaptive Load Balancing Algorithm Using Service Queue,” p. 4, 2012.

[25] K. D. Kumar and E. Umamaheswari, “Resource Provisioning in Cloud Computing Using Prediction Models: A Survey,” p. 10.

[26] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, “Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling,” 2016, pp. 1–10, doi: 10.1109/CLOUD.2016.0011.

[27] T. R. G. Nair and M. Vaidehi, “Efficient resource arbitration and allocation strategies in cloud computing through virtualization,” in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, 2011, pp. 397–401, doi: 10.1109/CCIS.2011.6045097.

[28] C. Jiang, X. Xu, J. Zhang, Y. Li, and J. Wan, “Resource Allocation in Contending Virtualized Environments through VM Performance Modeling and Feedback,” in *2011 Sixth Annual Chinagrid Conference*, 2011, pp. 196–203, doi: 10.1109/ChinaGrid.2011.44.

[29] M. H. Mohamaddiah, A. Abdullah, S. Subramaniam, and M. Hussin, “A Survey on Resource Allocation and Monitoring in Cloud Computing,” *Int. J. Mach. Learn. Comput.*, pp. 31–38, Feb. 2014, doi: 10.7763/IJMLC.2014.V4.382.

[30] R. Johnson, “EVALUATING THE USE OF SNMP AS A WIRELESS NETWORK MONITORING TOOL FOR IEEE 802.11 WIRELESS NETWORKS,” p. 117.

[31] L. Andrey, O. Festor, A. Lahmadi, A. Pras, and J. Schönwälder, “Survey of SNMP performance analysis studies: SURVEY OF SNMP PERFORMANCE ANALYSIS STUDIES,” *Int. J. Netw. Manag.*, vol. 19, no. 6, pp. 527–548, Nov. 2009, doi: 10.1002/nem.729.