

# COMPARISON AND PERFORMANCE EVALUATION OF SOA AND MICROSERVICES

*Submitted in partial fulfilment of the requirements of the degree of  
(Bachelor of Technology)  
by*

Anirudh Nanduri (167241)  
Anirudh Avire (167208)  
Avinash (167235)

## **Supervisor (s):**

Dr. S. Ravichandra  
Associate Professor, Department of CSE, NIT Warangal



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL**

(2020)

# APPROVAL SHEET

This Project Work entitled **Comparison and Performance Evaluation of SOA and Microservices** by **Anirudh Nanduri, Anirudh Avire, Avinash** is approved for the degree of Bachelor of Technology in Computer Science and Engineering at National Institute of Technology Warangal during the year 2019-2020

## Examiners

---

---

---

## Supervisor (s)

---

Dr. S. Ravichandra  
Associate Professor, CSE Department

## Chairman

---

Dr. P. Radha Krishna  
Head of Department, CSE  
NIT Warangal

Date:

Place:

# DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/ data/ fact/ source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

---

(Signature)

Anirudh Nanduri

167241

Date:

---

(Signature)

Anirudh Avire

167208

Date:

---

(Signature)

Avinash

167235

Date:

**NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

This is to certify that the Project work entitled **“COMPARISON AND PERFORMANCE EVALUATION OF SOA AND MICROSERVICES”** is a bonafide record of work carried out by Anirudh Nanduri (167241), Anirudh Avire (167208) and Avinash (167235), submitted to the faculty of Computer Science and Engineering Department in fulfillment of the requirements for the award of the degree of

Bachelor of Technology in Computer Science and Engineering at National Institute of Technology, Warangal during the academic year 2019-2020.

Dr. S. Ravichandra  
Project Guide  
Department of CSE  
NIT Warangal

Dr. P. Radha Krishna  
Head of Department  
Department of CSE  
NIT Warangal

## ACKNOWLEDGEMENT

We consider it as a great privilege to express our deep gratitude to many respected personalities who guided, inspired and helped us in the successful completion of our project.

We would like to express our deepest gratitude to our guide **Dr. S. Ravichandra** and **Vinay Raj**, Department of Computer Science and Engineering, National Institute of Technology, Warangal, for their constant supervision, guidance, suggestions and invaluable encouragement during the project. They have been a constant source of inspiration and helped in each stage.

We are grateful to **Dr. P. Radha Krishna**, Head of Department, Computer Science and Engineering, National Institute of Technology, Warangal for his moral support to carry out this project.

We are very thankful to the project evaluation committee for the strenuous efforts to evaluate our project.

We wish to thank all the staff members of the department for their kind cooperation and support given throughout our project work. We are also thankful to all our friends who have given their valuable suggestions and help in all stages of the development of the project.

Anirudh Nanduri (167241)

Anirudh Avire (167208)

Avinash (167235)

## ABSTRACT

Service-Oriented Architectures (SOA) is an architectural approach in which applications make use of services available in the network. SOA lends itself particularly well to model-driven implementation, because it is based on a high-level software module concept (the service) for which there are good definition and interface standards.

SOA development can be slow due to use of things like communication protocols SOAP, middleware and lack of principles. On the other hand, Microservices are agnostic to most of these things. Any technology stack, any hardware/middleware, any protocol can be used with Microservices as long as the principles of Microservices are followed.

The purpose of this report is to provide readers a solid understanding of Microservices and SOA by evaluating these architectures on the basis of their metrics. Evaluation of microservices is done on the various metrics such as load, number of instances, architectural patterns and effect on the response time has been determined.

We provide both research and industry perspectives to point out strengths and weaknesses of both architectural directions on the basis of software metrics. This report gives information for architects as to why choose Microservices architecture over web services. We have discussed metrics used for calculating Coupling between services and we evaluated by considering a retail vehicle sales application which is built using both the styles.

# CONTENTS

1. Introduction	1
1.1 Service Oriented Architecture (SOA)	1
1.2 Microservices	1
1.2.1 Pros Of Microservices	2
1.3 Eureka Server	2
1.4 Docker	3
1.4.1 Docker Image	4
1.4.2 Docker Container	4
1.5 JMeter	5
2. Related Work	7
3. Problem Statement	8
3.1 Metrics Related To Coupling	8
4. Implementation	10
4.1 Database Schema	11
4.2 Architecture	12
5. Experimental Results	14
5.1 Experimental Setup	14
5.1.1 Application Setup	14
5.1.2 Setup for Performance Testing	15
5.2 Understanding JMeter Metrics	16
5.3 Interpreting JMeter Metrics	16
5.4 Performance of Application in Microservice Architecture	17
5.5 Multiple Instances Of Microservices	22
5.5.1 Performance under multiple instances	23
5.6 Performance of Application in SOA	28
5.7 Comparison of Application Performance in SOA and Microservices	30
6. Conclusion	34
7. References	35

## **List of Figures**

1.1 Figure showing Monolithic, SOA, Microserices	2
1.2 Microservices Architecture	3
1.3 Docker	4
1.4 Docker Containers Vs Virtual Machines	5
1.5 Jmeter Dashboard	6
4.1 Database Schema of the Application	11
4.2 Design of the application in Microservice Architecture	12
4.3 Design of the application in SOA	14
4.4 Comparision on the basis of CS and RCS Values	15
5.1 Docker file for creating docker image	16
5.2 Jmeter configured for testing	17
5.3 CSV file generated by JMeter	19
5.4 Architecture of multiple instances	24



## **List of Tables**

4.1	List of services of the case study application along with CS and RCS values of microservices based application	13
4.2	List of services of the case study application along with CS and RCS values of SOA based application	14
5.1	: Response time of the microservices for 500 and 1000 users	24
5.2	: Response time for single and three instances	29
5.3	: Comparison of response times of application in Microservices and SOA	35

# CHAPTER 1

## INTRODUCTION

### 1.1 Service Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is an architectural approach in which applications make use of services available in the network. In this architecture, services are provided to form applications, through a communication call over the internet.

- SOA allows users to combine a large number of facilities from existing services to form applications.
- SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.
- SOA based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

There are two major roles within Service-oriented Architecture:

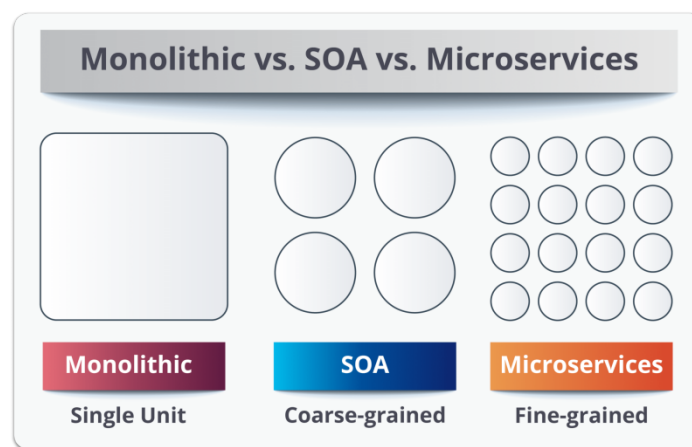
1. **Service provider:** The service provider is the maintainer of the service and the organization that makes available one or more services for others to use.
2. **Service consumer:** The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.

### 1.2 Microservices

Microservices, or microservice architecture, is an approach to application development in which a large application is built as a suite of modular components or services. Each module supports a specific task or business goal and uses a simple, well-defined interface, such as an application programming interface (API), to communicate with other sets of services. In microservice architecture, an application is divided into services. Each service runs a unique process and usually manages its own database. A service can generate alerts, log data; support user interfaces (UIs), handles user identification or authentication, and performs various other tasks.

### 1.2.1 Pros of Microservices

- Microservice architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team
- Code for different services can be written in different languages
- Easy to understand and modify for developers, thus can help a new team member become productive quickly
- Starts the web container more quickly, so the deployment is also faster
- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
- If one microservice fails, the other microservices will continue to work
- No long-term commitment to technology stack



*Fig 1.1 : Figure showing Monolithic, SOA, Microservices*

### 1.3 Eureka Server

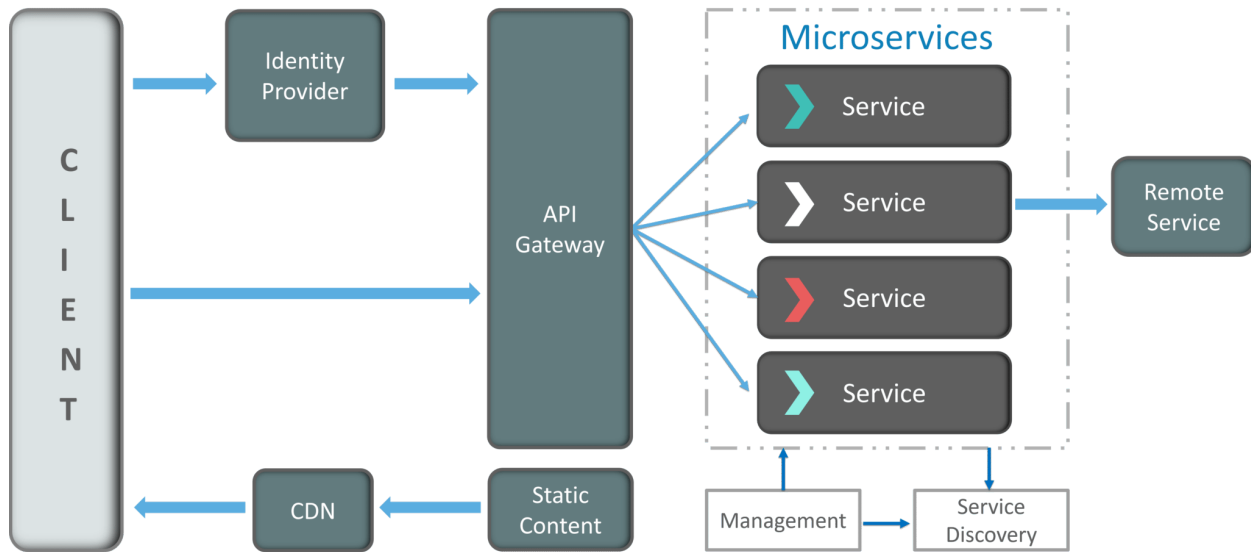
Eureka Server also known as Discovery Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. It acts as an API gateway.

Whenever a microservice A wants to communicate with another microservice B deployed on a different server, A should know the IP address and port of the server on which B is deployed. If the IP address of B is fixed, then you can use that approach, but what happens when your IP addresses and hostname are unpredictable?

Every microservice registers itself in the Eureka server when bootstrapped, generally using the {ServiceId} it registers into the Eureka server, or it can use the hostname or any public IP(if they are fixed)

After registering, every 30 seconds,it pings the Eureka server to notify it that the service itself is available. If the Eureka server is not getting any pings from a service for a quite long time, this service is unregistered from the Eureka server automatically and the Eureka server notifies the new state of the registry to all other services.

The Eureka server is nothing but another microservice which treats itself as a Eureka client.

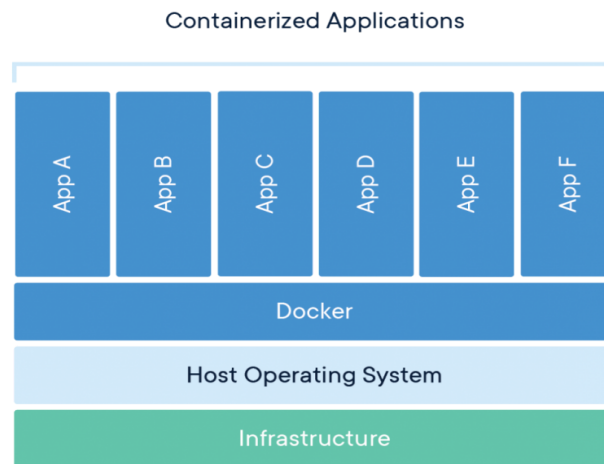


*Fig 1.2 : Microservices Architecture*

## 1.4 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications to be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.



*Fig 1.3 : Docker*

### 1.4.1 Docker Image

A Docker image is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

Due to their read-only quality, these images are sometimes referred to as snapshots. They represent an application and its virtual environment at a specific point in time. This consistency is one of the great features of Docker. It allows developers to test and experiment software in stable, uniform conditions.

Since images are, in a way, just templates, you cannot start or run them. What you can do is use that template as a base to build a container. A container is, ultimately, just a running image. Once you create a container, it adds a writable layer on top of the immutable image, meaning you can now modify it.

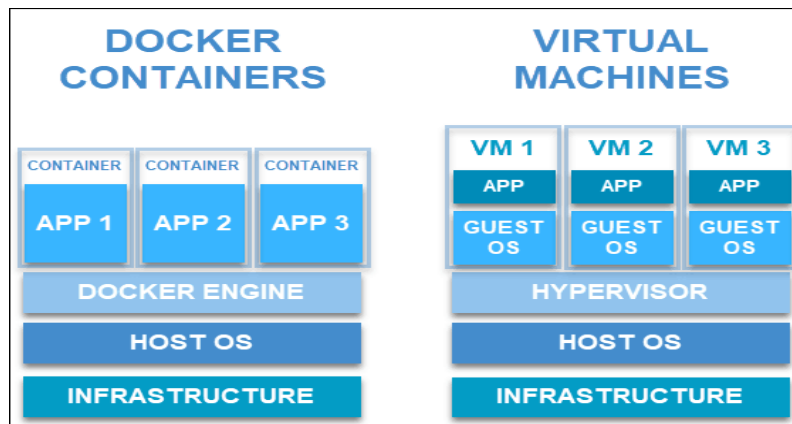
### 1.4.2 Docker Container

A Docker container is a virtualized run-time environment where users can isolate applications from the underlying system. These containers are compact, portable units in which you can start up an application quickly and easily.

A valuable feature is the standardization of the computing environment running inside the container. Not only does it ensure your application is working in identical circumstances, but it also simplifies sharing with other teammates.

As containers are autonomous, they provide strong isolation, ensuring they do not interrupt other running containers, as well as the server that supports them. Docker claims that these units “provide the strongest isolation capabilities in the industry”. Therefore, you won’t have to worry about keeping your machine secure while developing an application.

Unlike virtual machines (VMs) where virtualization happens at the hardware level, containers virtualize at the app layer. They can utilize one machine, share its kernel, and virtualize the operating system to run isolated processes. This makes containers extremely lightweight, allowing you to retain valuable resources.



*Fig 1.4 : Docker Containers Vs Virtual Machines*

## 1.5 JMeter

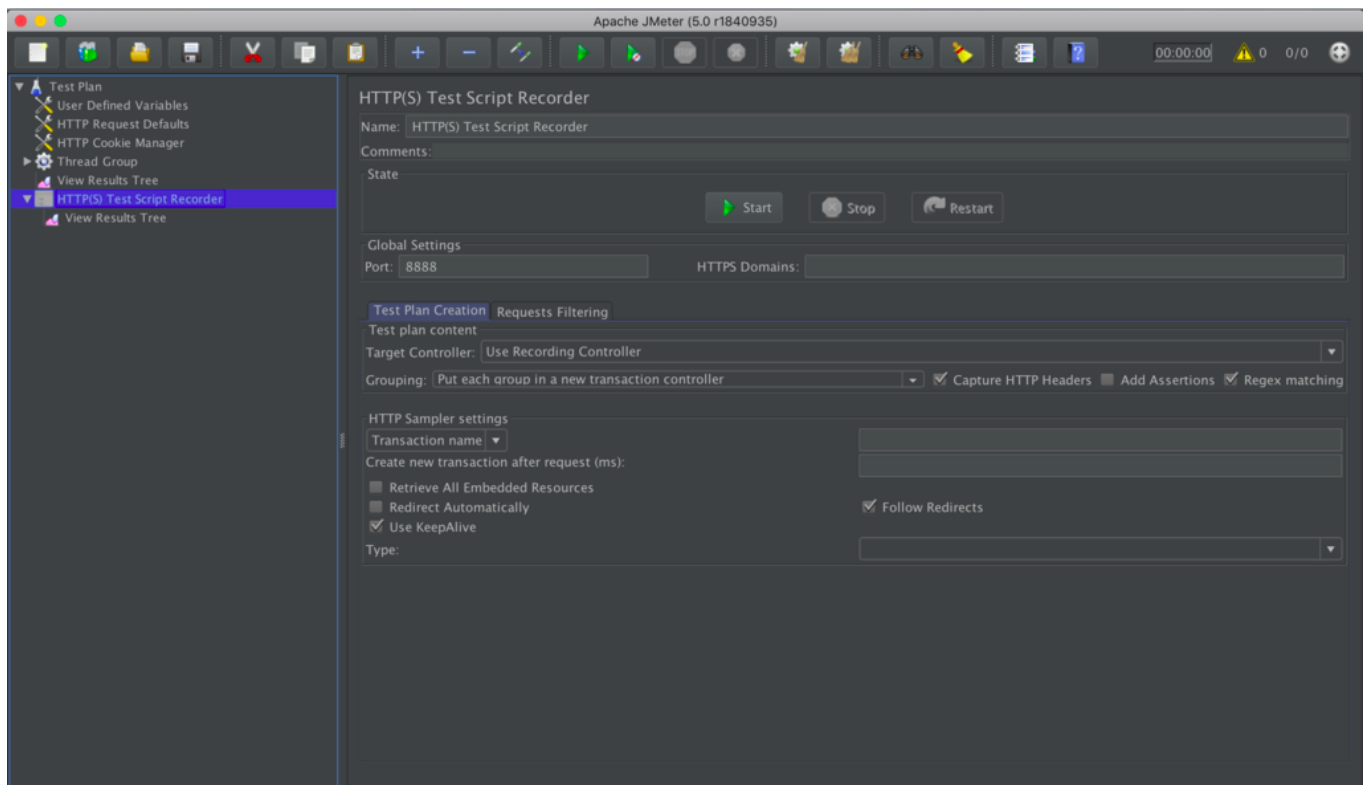
The Apache **JMeter™** application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.

Apache JMeter features include:

- Ability to load and performance test many different applications/server/protocol types:
  - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
  - SOAP / REST Web Services
  - FTP
  - Database via JDBC
  - LDAP
  - Message-oriented middleware (MOM) via JMS
  - Mail - SMTP(S), POP3(S) and IMAP(S)
  - Native commands or shell scripts
  - TCP
  - Java Objects

- Full featured Test IDE that allows fast Test Plan recording (from Browsers or native applications), building and debugging
- A complete and ready to present dynamic HTML report
- Easy correlation through ability to extract data from most popular response formats, HTML, JSON , XML or any textual format
- Complete portability and 100% Java purity
- Full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups
- Caching and offline analysis/replaying of test results



*Fig 1.5 : Jmeter Dashboard*

## CHAPTER 2

### RELATED WORK

A systematic mapping study conducted by Taibi D *et al.* stated the open issues and research gaps in the emerging topic of microservices architecture [1]. Among the emerging issues highlighted, comparison of both service oriented architecture and microservices is one which we have selected. The authors state that there is a lack of comparison between these two architectures in terms of performance, coupling , development effort and maintenance. In this work, we choose to compare both the styles with loose coupling first and then we compare them in terms of performance.

In this chapter, we will discuss different criteria under which the application and two architectures are compared.

***Metrics related to coupling:*** As a first step, we have calculated the coupling related metrics of the application in microservices architecture.

***Performance of Microservices Application under different load:*** In the first step of the evaluation, we have considered the microservice application performance under a load of 500 and 100 users.

***Performance improvement under multiple instances:*** As a second part of evaluation, we have considered testing the microservice application under the same load but multiple instances.

***Performance of application in SOA:*** We have built the same application in SOA and obtained the performance metrics.

***Comparison of performance in Microservice architecture and SOA:*** As the final part of evaluation, both the architectures are compared from the performance metrics under a load of users.



# CHAPTER 3

## PROBLEM STATEMENT

There is an industrial shift from Service-Oriented Architectures (SOA) into Microservices. Software architects are still hesitant to adopt Microservices architecture despite its benefits over SOA based web services because they are not fully aware of the pros and cons. This work of ours is an attempt to help software architects as to why choose Microservices over web services.

In this work, first we theoretically analyze the coupling between the services of both the architectures using the software metrics. Later, the performance of an application built using both the architectural styles will be compared.

The purpose of this project is to verify the feasibility of the migration of SOA to Microservices architecture. An application is implemented in both the architectures and software metrics have been calculated to compare the performance.

Response time of the Microservice application has been evaluated to understand the performance in different conditions.

### 3.1 Metrics Related To Coupling

As Coupling is the most important feature of SOA based systems we have chosen metrics of coupling. Services in service oriented systems should have less coupling, Services should be loosely coupled.

**Number of Services(NoS):** This is the basic metric which gives the count of services in the architecture. This metric is used in calculating other complexity metrics.

$$NoS = |S[*]|$$

where  $S[*]$  is the count of services present at all nodes

**Coupling of Services(CS):** Coupling of Services metric is given by the number of interactions. Each service has with other services for performing the desired operations is group of services provided by the node.

$$CS[s] = |\Pi_{\text{service } \sigma \text{ invoker} = s} (A)|$$

Where  $S[n]$  is group of services provided by the node  $N$  is the group of service provider's nodes.

$A$  is set of tuples

***Relative Coupling of Service(RCS)***: RCS denotes the degree of coupling in a particular service. RCS of a service is denoted by

$$RCS[s] = \frac{cs[s]}{NoS}$$

This metric has more impact in calculating the complexity of the system. If the

RCS value of the service is less, it means service has low coupling and if the RCS value is high it indicates that service has high coupling value

# CHAPTER 4

## IMPLEMENTATION

We have considered a **Retail Vehicle** application [11] which is built using SOA and Microservices separately. XYZ is a retail Web-based application that allows consumers to choose, customize, and locate new and pre-owned vehicles using a fast, secure, and easy-to-use Web interface. The purpose of this Web site is to increase sales by providing customers with the functionality to choose, customize, compare vehicles, locate a dealer, and request a quote. The Web site provides pricing information about the base models and the various options and accessories that customers can choose based on their budget. The Web site also provides real-time incentives data, and shows configured vehicle prices with the incentives applied. Using the Web site, customers can search for dealers in their local area and also view the inventory data of each of these dealers.

The application has been built in microservices architecture using Spring Boot. Eureka server has been used as Service Registry. Microservices communicate between each other using REST API and respond to requests.

We had JDK 11, Spring Boot , Maven and another half a dozen libraries of the Spring Cloud stack for service discovery, client-side load balancing.

MYSQL is used as a backend database. It is connected to Spring Boot application using JPA connector. The application is containerised using Docker for portability. Each service application is containerised and load is given to the docker container to measure performance.

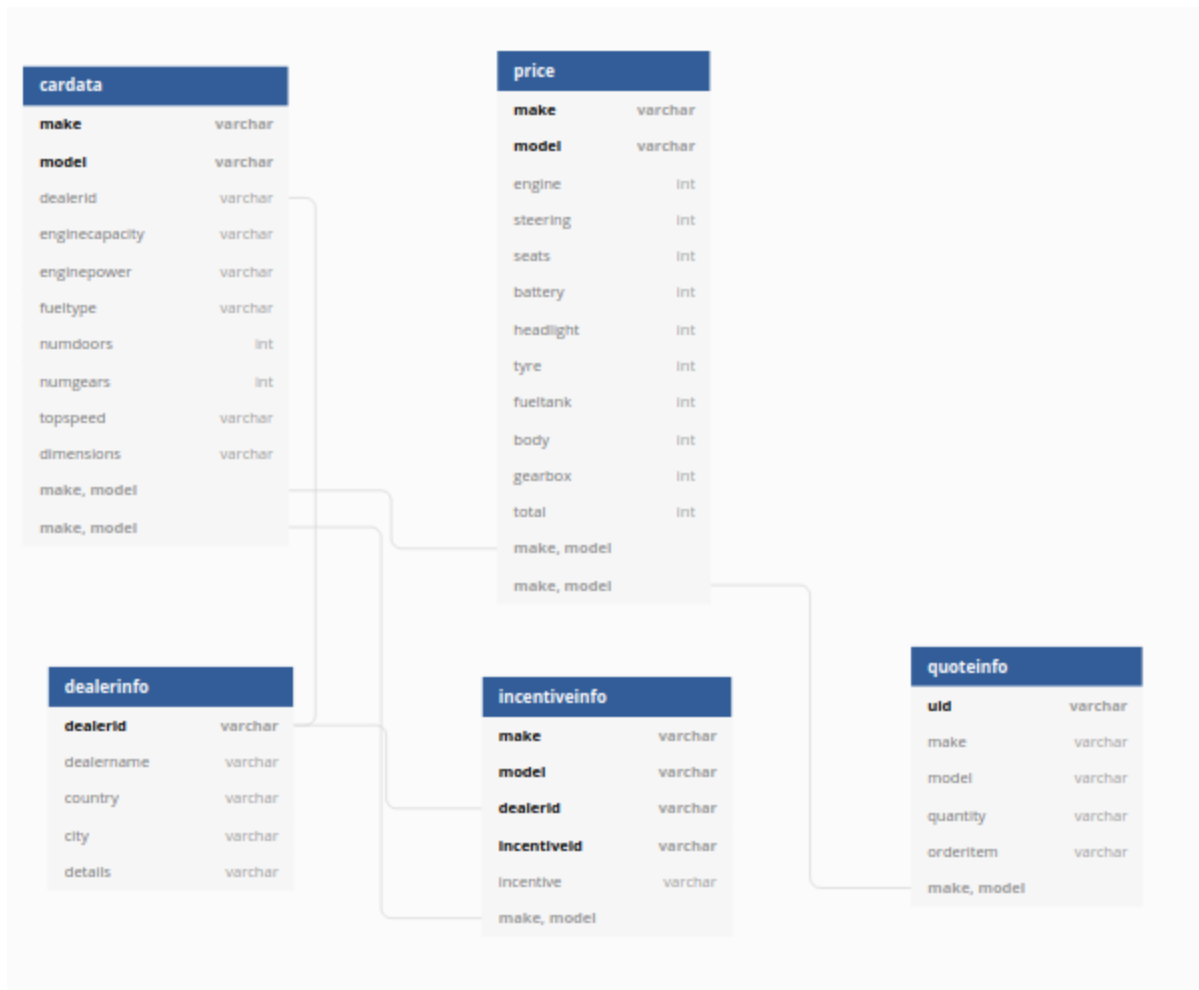
While these technologies continue to grow in popularity, we see some new trends on the horizon: 1) test automation, 2) Continuous Deployment/Verification (CD/CV), 3) incident response, 4) Cloud Service Expense Management (CSEM)

The codebase of the application can be found at <https://github.com/Nanirudh/RetailVehicleApp.git>

## 4.1 Database Schema

The database consists of five tables. Spring Boot connector retrieves data from MYSQL database using JPA connector.

- **Cardata** table stores data about the car parts
- **Price** table stores data about the price of car parts
- **Dealerinfo** table stores data about car dealers
- **Incentiveinfo** table stores data about the incentives offered by dealers for the purchase of car and its parts
- **QuoteInfo** table stores data about the quote given by dealers when the customers provide their details

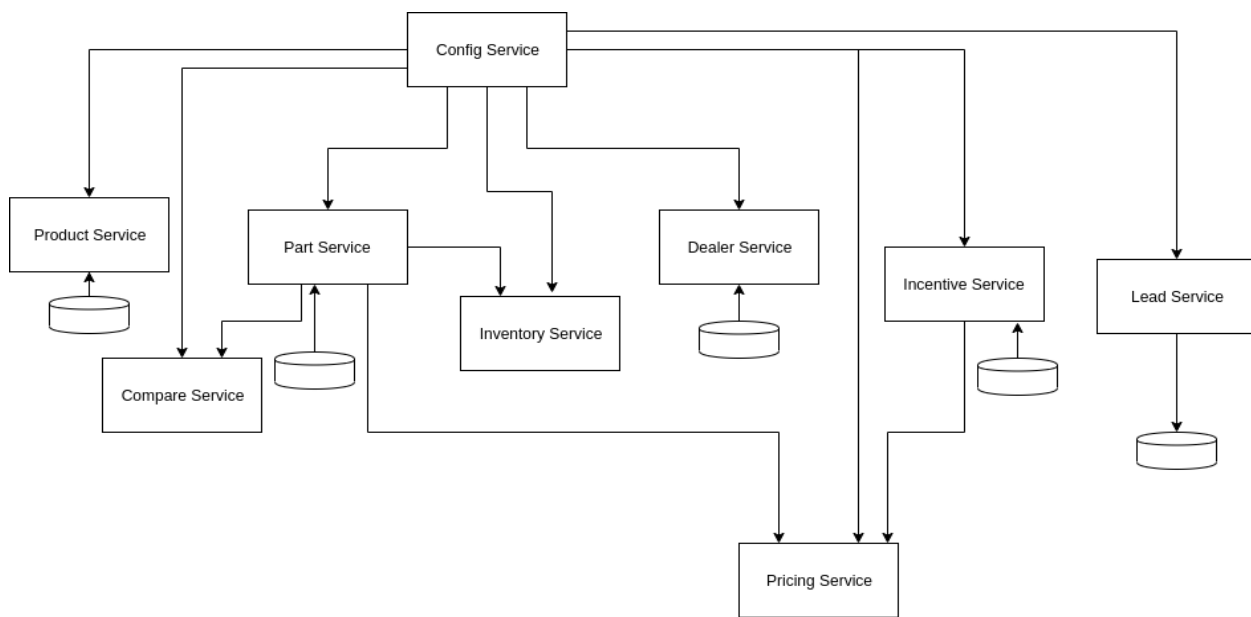


*Fig 4.1 : Database Schema of the Application*

## 4.2 Architecture

The figure below gives an overview about the microservices which interact with each other. Config Service is the entry point microservice where the service request is made. Config Service interacts with service registry eureka microservice and retrieves the appropriate response.

Part Service, Product Service, Dealer Service, Incentive Service and Lead Service are connected to their MYSQL database using JPA connector.



*Fig 4.2 : Design of the application in microservice architecture*

Below is the information on microservices interaction. Based on this information, CS value and RCS values are calculated.

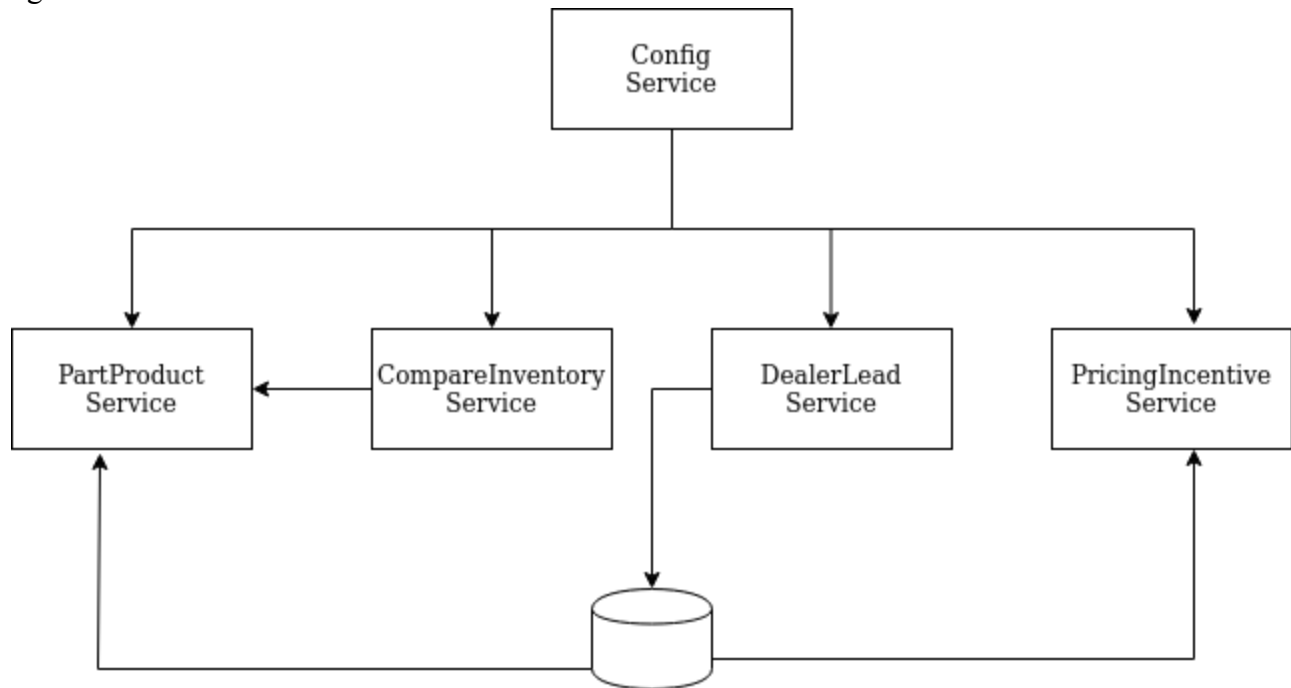
- Eureka Server acts as Service Registry.
- Config Service interacts with Product Service, Compare Service, Inventory Service, Lead Service, Part Service, Incentive Service, Dealer Service, Pricing Service.
- Part Service interacts with Compare Service, Inventory Service ,Config Service.
- Product Service interacts with Config Service.
- Compare Service interacts with Part Service and Config Service.
- Incentive Service interacts with Pricing Service and Config Service.
- Pricing Service interacts with Incentive Service and Config Service.
- Inventory Service interacts with Part Service and Config Service.
- Lead Service interacts with Dealer Service and Config Service.

Microservice #	Microservice Name	Interacting Service #	CS Value	RCS Value
1	Config Service	2,3,4,5,6,7,8,9	8	0.88
2	Part Service	1,4,8	3	0.33
3	Product Service	1	1	0.11
4	Compare Service	1,2	2	0.22
5	Incentive Service	1,6	2	0.22
6	Pricing Service	1,5	2	0.22
7	Dealer Service	1,9	2	0.22
8	Inventory Service	1,2	2	0.22
9	Lead Service	1,7	2	0.22

*Table 4.1 : Table shows the list of services of the case study application along with CS and RCS values of microservices based application*

The figure below gives an overview about the services in SOA and their interaction with each other. Config Service is the entry point microservice where the service request is made. Config Service interacts with service registry eureka service and retrieves the appropriate response.

PartProduct Service, DealerLead Service, PricingIncentive Service are connected to MYSQL database using JPA connector.



*Fig 4.3 : Design of the application in SOA*

Below is the information on services interaction in SOA. Based on this information, CS values and RCS values are calculated.

- Eureka Server acts as Service Registry.
- Config Service interacts with PartProduct Service, CompareInventory Service, DealerLead Service, PricingIncentive Service.
- PartProduct Service interacts with Compare Inventory Service ,Config Service.
- CompareInventory Service interacts with PartProduct Service and Config Service.
- PricingIncentive Service interacts with Config Service.
- DealerLead Service interacts with Config Service.

Service#	Service Name	Interacting Service#	CS Value	RCS Value
1	Config Service	2,3,4,5	4	0.8
2	PartProduct Service	1,2,5	3	0.6
3	PricingIncentive Service	1	1	0.2
4	DealerLead Service	1	1	0.2
5	CompareInventory Service	1,2	2	0.4

Table 4.2 : Table shows the list of services of the case study application along with CS and RCS values of SOA

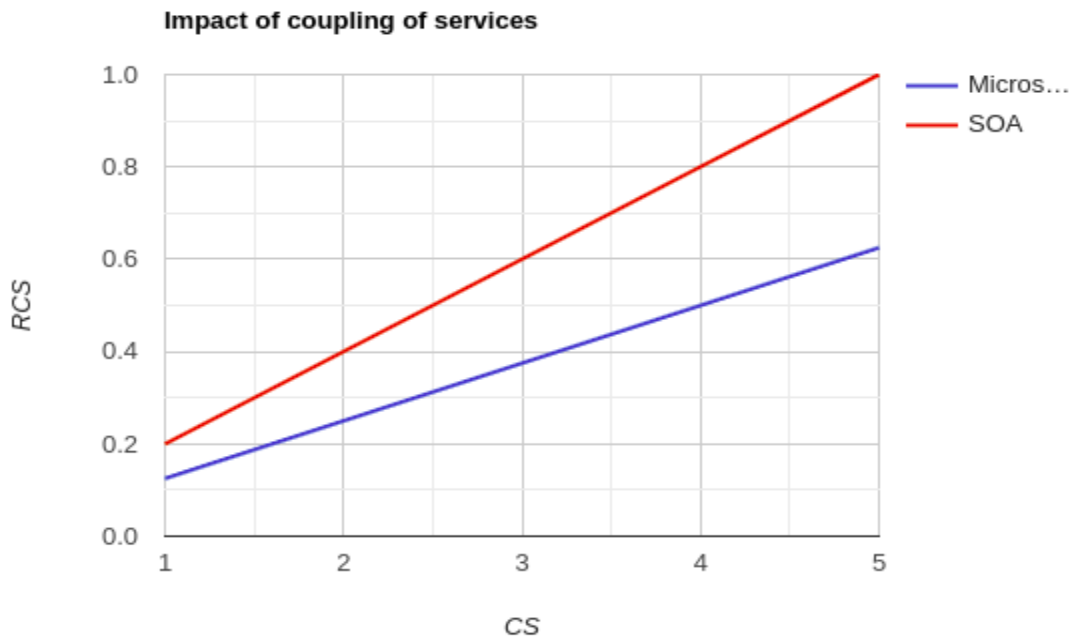


Fig 4.4 : Comparison on the basis of CS and RCS Values

The coupling values of both the styles are calculated and compared. We have plotted the RCS and CS values in a graph with X axis representing the Coupling of Services(CS) values and Y-axis representing the Relative Coupling of Services(RCS) values. Architecture is said to be good if it has low coupling value among its services. From the graph, it is clear that the Relative Coupling of Services(RCS) values of Microservice architecture are less. It is also observed that Microservices style has lower RCS values compared to Web Service style for each CS value. Hence it motivates to choose Microservices architecture style for designing enterprise applications. Similar kinds of comparison can be made using other architectural metrics like Relative Importance of Service, Network Cohesion in the system etc.



## EXPERIMENTAL RESULTS

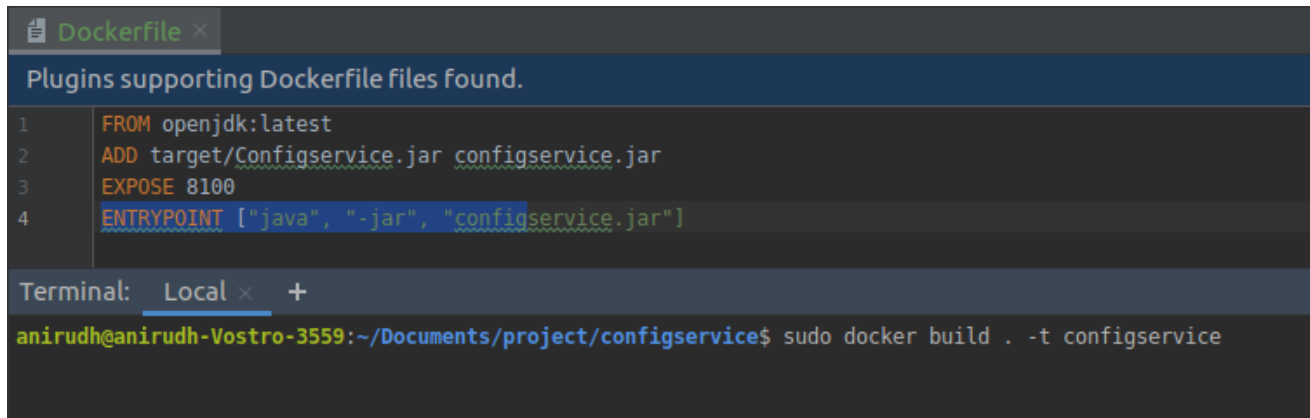
## 5.1 Experimental Setup

## 5.1.1 Application Setup

Docker image of microservices can be accessed from

<https://hub.docker.com/search?q=aaaproject&type=image>

Docker image of the JAR file of the spring boot project is created.



The screenshot shows a Dockerfile with the following content:

```

1 FROM openjdk:latest
2 ADD target/Configservice.jar configservice.jar
3 EXPOSE 8100
4 ENTRYPOINT ["java", "-jar", "configservice.jar"]

```

Below the Dockerfile, the terminal output shows the command to build the Docker image:

```

Terminal: Local x +
anirudh@anirudh-Vostro-3559:~/Documents/project/configservice$ sudo docker build . -t configservice

```

*Fig 5.1 : Docker file for creating docker image*

Docker containers are created from docker images using the following commands

- `sudo docker run --name mysql-db -e MYSQL_ROOT_PASSWORD=MyDb123 -e MYSQL_DATABASE=DB -e MYSQL_USER=anirudh -e MYSQL_PASSWORD=MyDb123 -d mysql`
- `sudo docker run -p 8761:8761 --name eurekaservice -d eurekaservice`
- `sudo docker run -p 8106:8106 --name partservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d partservice`
- `sudo docker run -p 8108:8108 --name productservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d productservice`
- `sudo docker run -p 8107:8107 --name pricingservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d pricingservice`
- `sudo docker run -p 8102:8102 --name dealerservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d dealerservice`
- `sudo docker run -p 8103:8103 --name incentiveservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d incentiveservice`
- `sudo docker run -p 8105:8105 --name leadservice --link mysql-db:mysql --link eurekaservice:eurekaservice -d leadservice`
- `sudo docker run -p 8101:8101 --name comparservice --link mysql-db:mysql --link eurekaservice:eurekaservice --link partservice:partservice -d comparservice`

- `sudo docker run -p 8104:8104 --name inventoryservice --link mysql-db:mysql --link eurekaservice:eurekaservice --link partservice:partservice -d inventoryservice`
- `sudo docker run -p 8100:8100 --name configservice --link mysql-db:mysql --link eurekaservice:eurekaservice --link partservice:partservice --link productservice:productservice --link compareservice:compareservice --link inventoryservice:inventoryservice --link pricingservice:pricingservice --link dealerservice:dealerservice --link incentiveservice:incentiveservice --link leadservice:leadservice -d configservice`

### 5.1.2 Setup for Performance Testing

- JMeter Application is used for performance testing of the microservices

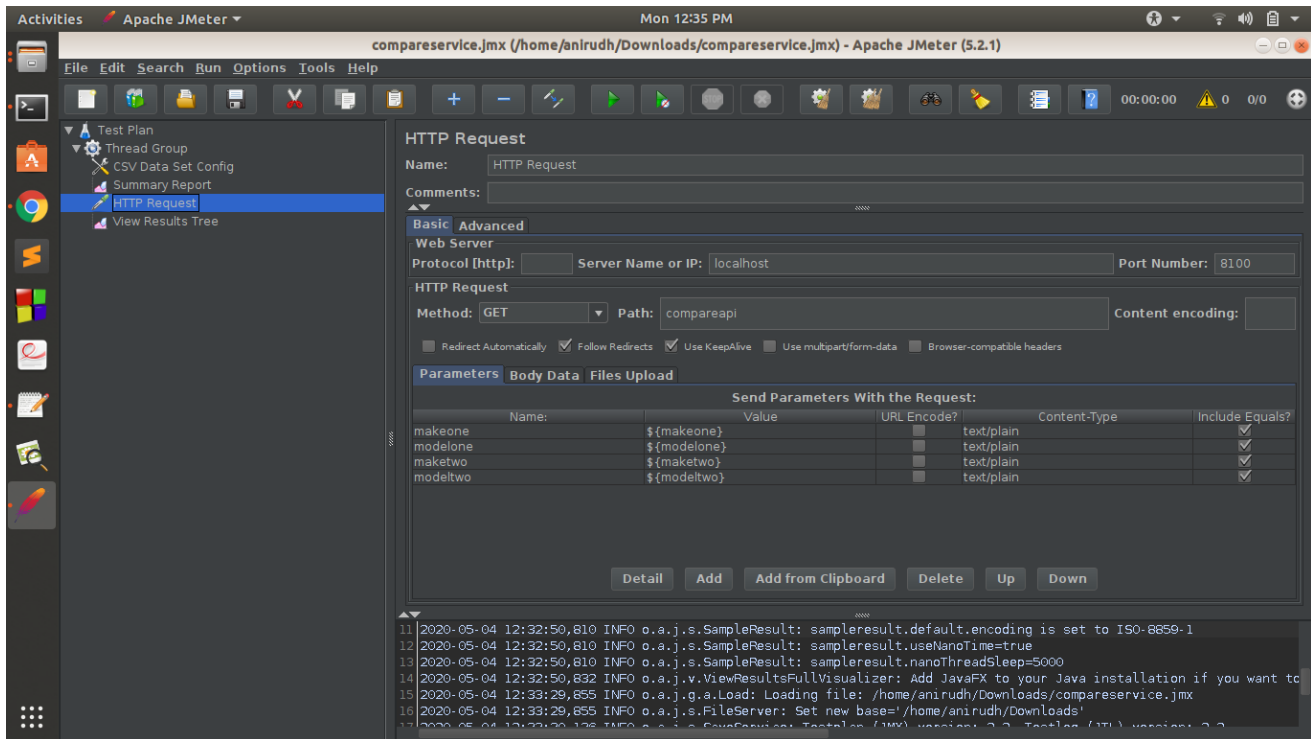


Fig 5.2 : Jmeter configured for testing

```
anirudh@anirudh-Vostro-3559:~/Documents/apache-jmeter-5.2.1 (1)/apache-jmeter-5.2.1/bin$ ./jmeter -n -t /home/anirudh/Documents/loadtest/partserviceoneinstance.jmx -l /home/anirudh/Documents/apache-jmeter-5.2.1/bin/results/partone.csv
```

The Microservices are tested under a load of 500 users(yellow line) and 1000 users(blue line) and their response time results are mentioned below. Here each user corresponds to a thread.

## 5.2 Understanding JMeter Metrics

**Response time** : JMeter measures the latency from just before sending the request to just after the first **response** has been received. Thus the **time** includes all the processing needed to assemble the request as well as assembling the first part of the **response**, which in general will be longer than one byte.

**Elapsed time**: Measures the elapsed time from just before sending the request to just after the last chunk of the response has been received.

**Latency**: Measures the latency from just before sending the request to just after the first chunk of the response has been received.

**Connect Time**: Measures the time it took to establish the connection, including SSL handshake.

**Median**: Number which divides the samples into two equal halves.

**90% Line (90th Percentile)**: The elapsed time below which 90% of the samples fall.

**Standard Deviation**: Measure of the variability of a data set. This is a standard statistical measure,

**Thread Name**: Derived from the Thread Group name and the thread within the group. The name has the format groupName + " " + groupIndex + "-" + threadIndex where:

**groupName**: name of the Thread Group element,

**groupIndex**: number of the Thread Group in the Test Plan, starting from 1,

**threadIndex**: number of the thread within the Thread Group, starting from 1.

**Throughput**: Calculated as requests/unit of time. The time is calculated from the start of the first sample to the end of the last sample. The formula is:  $\text{Throughput} = (\text{number of requests}) / (\text{total time})$ .

## 5.3 Interpreting JMeter Metrics

How do you know if a metric is satisfying or awful? Here are some explanations:

**Elapsed Time / Connect Time / Latency**: should be as low as possible, ideally less than 1 second. Amazon found every 100ms costs them 1% in sales, which translates to several millions of dollars lost,

**Median**: should be close to average elapsed response time,

**XX% line**: should be as low as possible too. When it's way lower than average elapsed time, it indicates that the last XX% requests have dramatically higher response times than lower ones,

**Standard Deviation**: should be low. A high deviation indicates discrepancies in responses times, which translates to response time spikes.

## 5.4 Performance of the Application in Microservice Architecture

The application for which the performance evaluation is being done is Retail Vehicle Application discussed above. Retail Vehicle application allows consumers to choose, customize, and locate new and pre-owned vehicles using a fast, secure, and easy-to-use Web interface. The purpose of this Web site is to increase sales by providing customers with the functionality to choose, customize, compare vehicles, locate a dealer, and request a quote.

Load is given to each microservice using JMeter testing application. CSV file depicting the performance metrics is obtained as a response.

The following reports are generated from CSV file using JMETER command

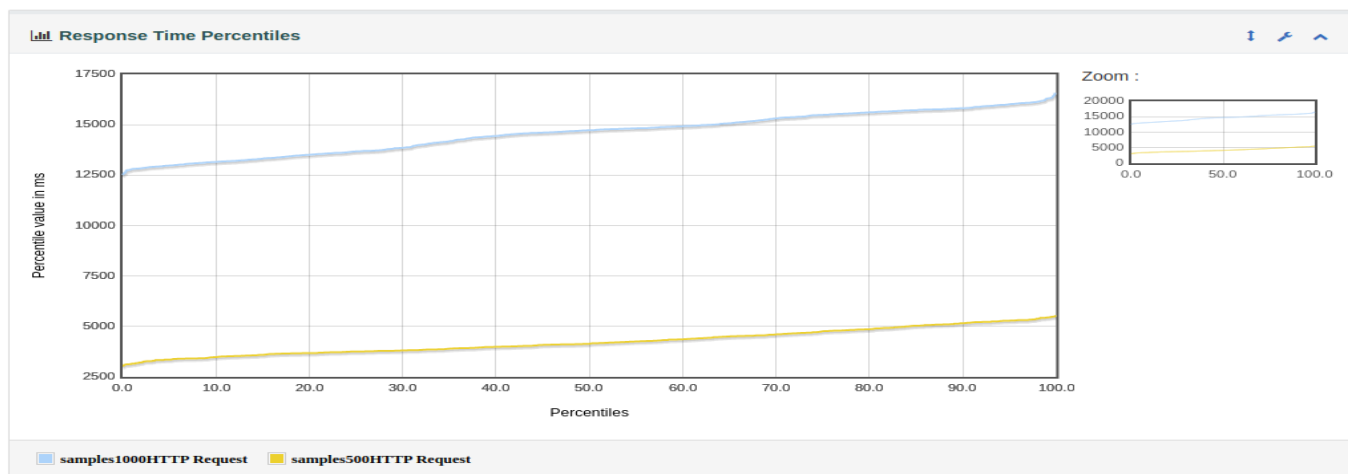
jmeter -g <csv file> -o <Path to output folder>

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	timeStamp	elapsed	label	responseCode	responseMessage	threadName	dataType	success	failure	bytes	sentBytes	grpThreads	allThreads	URL	Latency	IdleTime	Connect
2	1586191003535	14	HTTP Request	200		Thread Group 1-?text	text	true		581	209	1	1	http://localhost:8100/compareapi?mak	13	0	0
3	1586191003549	11	HTTP Request	200		Thread Group 1-?text	text	true		596	230	1	1	http://localhost:8100/compareapi?ma	11	0	0
4	1586191003560	7	HTTP Request	200		Thread Group 1-?text	text	true		572	198	1	1	http://localhost:8100/compareapi?mak	7	0	0
5	1586191003567	6	HTTP Request	200		Thread Group 1-?text	text	true		587	219	1	1	http://localhost:8100/compareapi?ma	6	0	0
6	1586191003573	6	HTTP Request	200		Thread Group 1-?text	text	true		599	219	1	1	http://localhost:8100/compareapi?mak	6	0	0
7	1586191003579	8	HTTP Request	200		Thread Group 1-?text	text	true		614	240	1	1	http://localhost:8100/compareapi?ma	8	0	0
8	1586191003587	6	HTTP Request	200		Thread Group 1-?text	text	true		603	223	1	1	http://localhost:8100/compareapi?mak	6	0	0
9	1586191003593	8	HTTP Request	200		Thread Group 1-?text	text	true		618	244	1	1	http://localhost:8100/compareapi?mak	8	0	0
10	1586191003601	6	HTTP Request	200		Thread Group 1-?text	text	true		585	203	1	1	http://localhost:8100/compareapi?ma	6	0	0
11	1586191003607	5	HTTP Request	200		Thread Group 1-?text	text	true		602	225	1	1	http://localhost:8100/compareapi?ma	5	0	0
12	1586191003613	5	HTTP Request	200		Thread Group 1-?text	text	true		587	204	1	1	http://localhost:8100/compareapi?ma	5	0	0
13	1586191003618	11	HTTP Request	200		Thread Group 1-?text	text	true		601	225	1	1	http://localhost:8100/compareapi?ma	11	0	0
14	1586191003629	6	HTTP Request	200		Thread Group 1-?text	text	true		594	214	1	1	http://localhost:8100/compareapi?ma	6	0	0
15	1586191003635	11	HTTP Request	200		Thread Group 1-?text	text	true		609	235	1	1	http://localhost:8100/compareapi?ma	11	0	0
16	1586191003646	8	HTTP Request	200		Thread Group 1-?text	text	true		610	232	1	1	http://localhost:8100/compareapi?ma	8	0	0
17	1586191003654	7	HTTP Request	200		Thread Group 1-?text	text	true		625	253	1	1	http://localhost:8100/compareapi?ma	7	0	0
18	1586191003661	7	HTTP Request	200		Thread Group 1-?text	text	true		604	226	1	1	http://localhost:8100/compareapi?ma	7	0	0
19	1586191003668	8	HTTP Request	200		Thread Group 1-?text	text	true		615	253	2	2	http://localhost:8100/compareapi?ma	8	0	0
20	1586191003674	11	HTTP Request	200		Thread Group 1-?text	text	true		635	242	2	2	http://localhost:8100/compareapi?ma	11	0	0
21	1586191003676	11	HTTP Request	200		Thread Group 1-?text	text	true		645	256	2	2	http://localhost:8100/compareapi?ma	11	0	0
22	1586191003685	8	HTTP Request	200		Thread Group 1-?text	text	true		634	242	2	2	http://localhost:8100/compareapi?m	8	0	0
23	1586191003687	7	HTTP Request	200		Thread Group 1-?text	text	true		642	252	2	2	http://localhost:8100/compareapi?m	7	0	0
24	1586191003693	7	HTTP Request	200		Thread Group 1-?text	text	true		636	242	2	2	http://localhost:8100/compareapi?ma	7	0	0
25	1586191003694	8	HTTP Request	200		Thread Group 1-?text	text	true		644	254	2	2	http://localhost:8100/compareapi?ma	8	0	0
26	1586191003700	6	HTTP Request	200		Thread Group 1-?text	text	true		632	238	2	2	http://localhost:8100/compareapi?m	6	0	0
27	1586191003702	6	HTTP Request	200		Thread Group 1-?text	text	true		640	250	2	2	http://localhost:8100/compareapi?m	6	0	0
28	1586191003706	8	HTTP Request	200		Thread Group 1-?text	text	true		636	242	2	2	http://localhost:8100/compareapi?ma	8	0	0
29	1586191003708	11	HTTP Request	200		Thread Group 1-?text	text	true		669	285	2	2	http://localhost:8100/compareapi?ma	10	0	0
30	1586191003714	8	HTTP Request	200		Thread Group 1-?text	text	true		647	255	2	2	http://localhost:8100/compareapi?m	8	0	0
31	1586191003719	8	HTTP Request	200		Thread Group 1-?text	text	true		657	269	2	2	http://localhost:8100/compareapi?m	7	0	0
32	1586191003722	6	HTTP Request	200		Thread Group 1-?text	text	true		653	261	2	2	http://localhost:8100/compareapi?ma	6	0	0
33	1586191003727	6	HTTP Request	200		Thread Group 1-?text	text	true		663	275	2	2	http://localhost:8100/compareapi?ma	6	0	0
34	1586191003728	10	HTTP Request	200		Thread Group 1-?text	text	true		621	225	2	2	http://localhost:8100/compareapi?m	10	0	0
35	1586191003734	7	HTTP Request	200		Thread Group 1-?text	text	true		633	241	2	2	http://localhost:8100/compareapi?m	7	0	0
36	1586191003738	7	HTTP Request	200		Thread Group 1-?text	text	true		638	246	2	2	http://localhost:8100/compareapi?m	7	0	0

Fig 5.3 : CSV File generated by JMeter

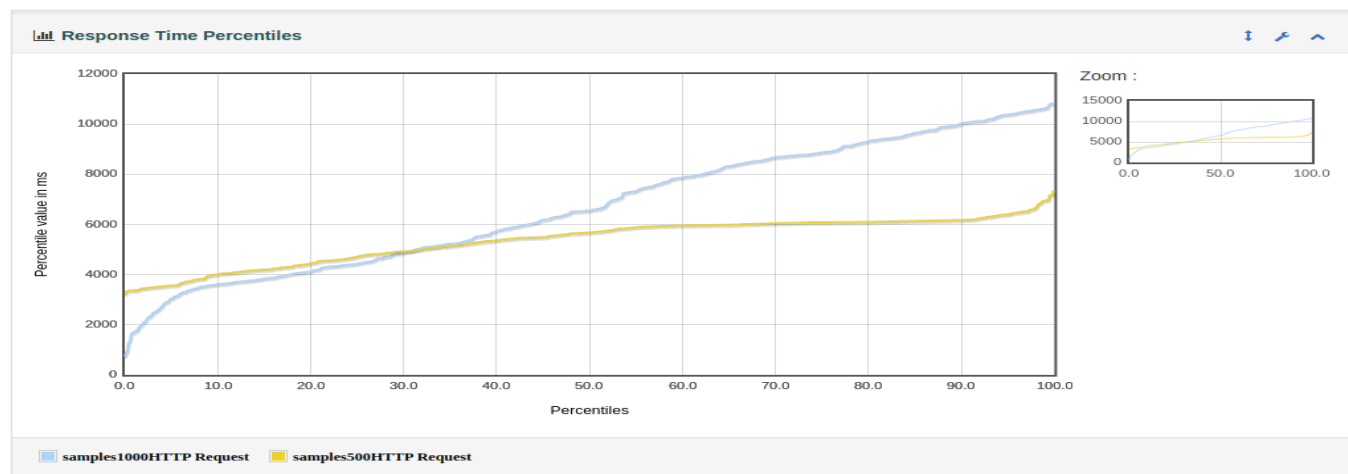
The following figures show the performance metrics of each microservice comparing the response time of the microservice with 500 threads(users) and 1000 threads(users)

## Part Microservice



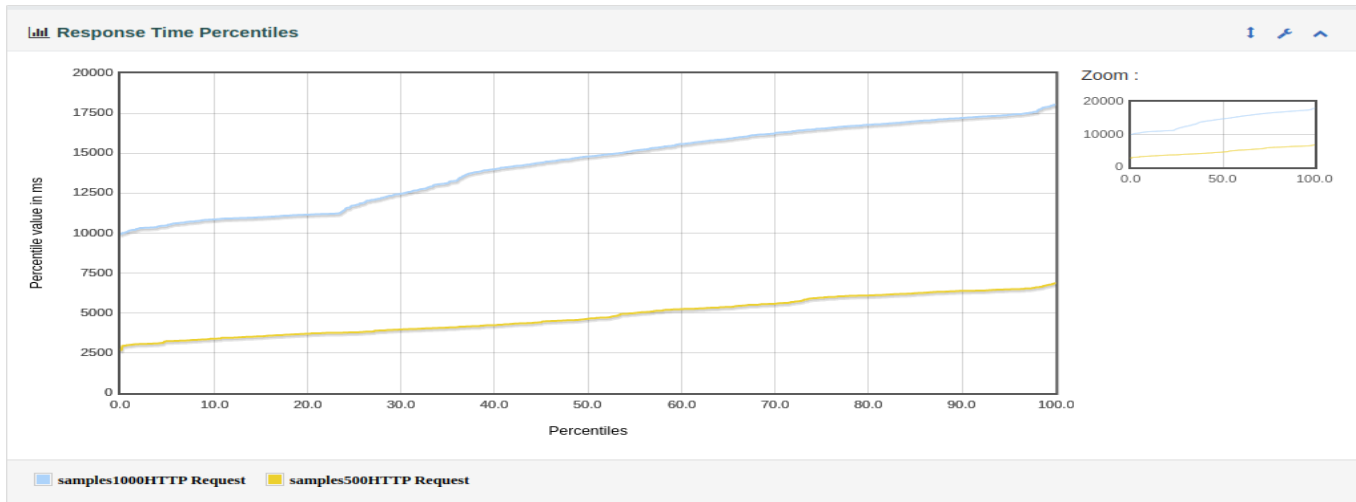
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	7	0.47%	11142.60	3049	16586	15721.80	15926.75	16189.88	81.48	32.45	14.99
samples500HTTP Request	500	0	0.00%	4245.85	3049	5514	5164.20	5288.85	5448.94	84.09	33.41	15.45
samples1000HTTP Request	1000	7	0.70%	14590.98	12511	16586	15819.50	16020.50	16300.71	55.43	22.10	10.20

## Product Microservice



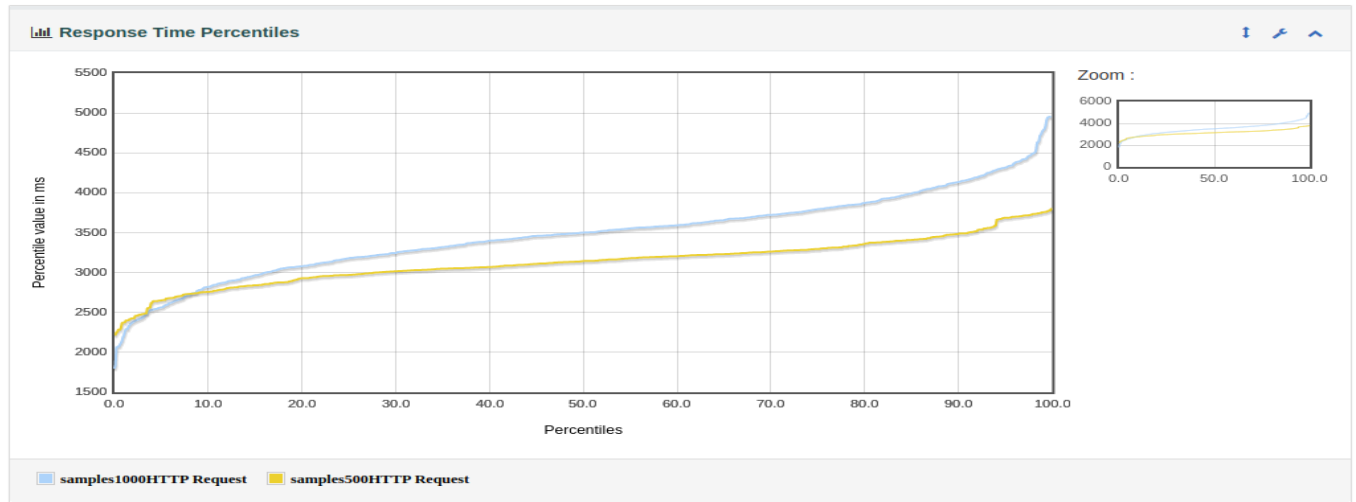
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	0	0.00%	6268.68	737	10858	9655.10	10151.40	10608.98	94.06	2741.32	12.99
samples500HTTP Request	500	0	0.00%	5381.52	3244	7335	6183.90	6404.00	6967.76	62.34	1824.70	8.61
samples1000HTTP Request	1000	0	0.00%	6712.26	737	10858	10022.00	10384.95	10656.69	62.70	1823.59	8.66

# Compare Microservice



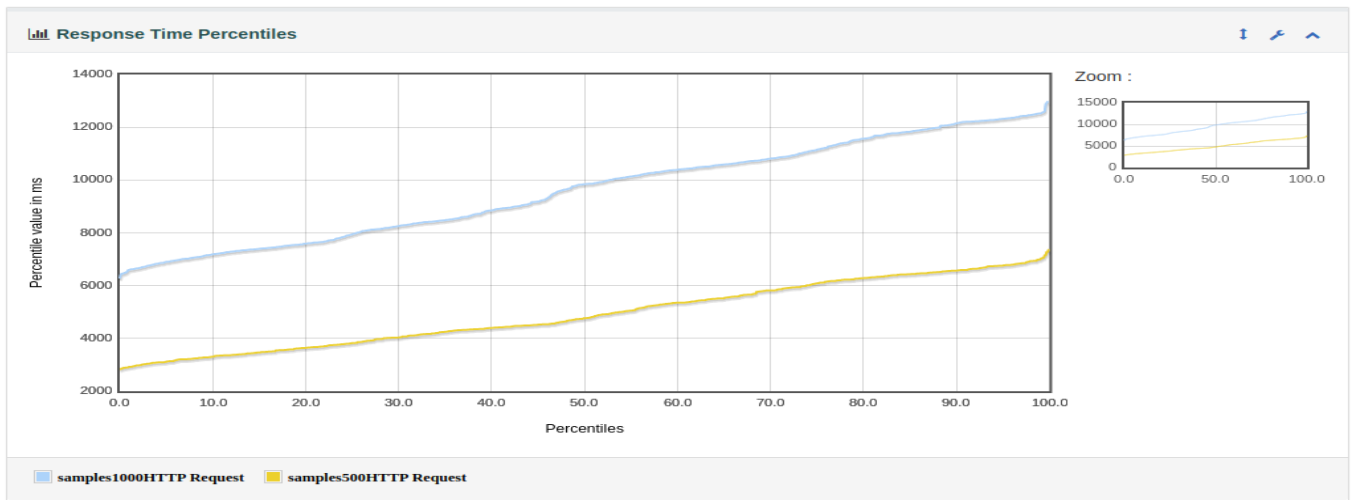
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	7	0.47%	11137.11	2691	18086	16997.70	17298.95	17778.78	66.27	42.40	18.04
samples500HTTP Request	500	0	0.00%	4818.51	2691	6863	6390.00	6491.00	6757.69	65.45	41.95	17.14
samples1000HTTP Request	1000	7	0.70%	14296.40	9951	18086	17212.50	17401.85	17884.91	44.22	28.27	12.27

# Pricing Microservice



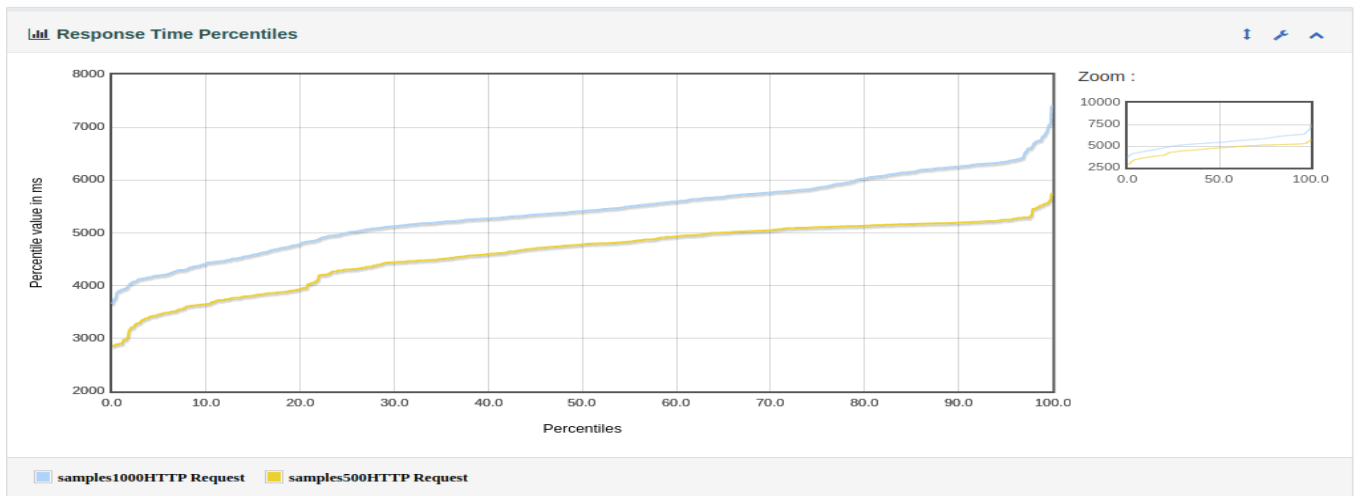
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	7	0.47%	3365.40	1795	4955	3989.70	4212.00	4632.98	176.33	67.65	32.96
samples500HTTP Request	500	0	0.00%	3133.11	2220	3798	3489.30	3687.00	3756.99	108.70	41.59	20.29
samples1000HTTP Request	1000	7	0.70%	3481.54	1795	4955	4135.60	4316.80	4778.75	117.55	45.16	21.99

# Inventory Microservice



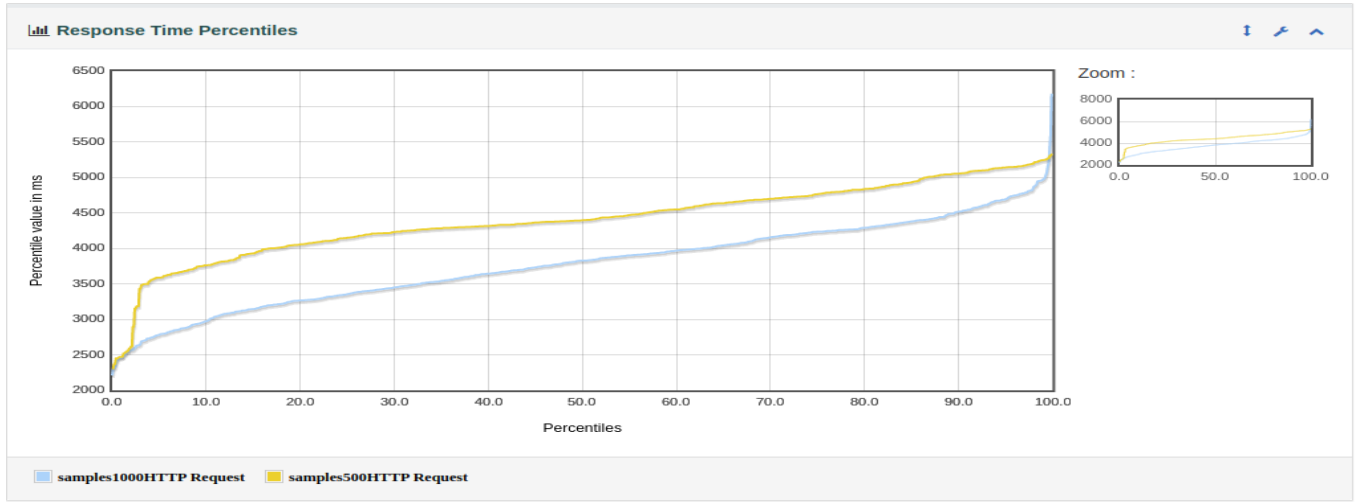
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	0	0.00%	8043.65	2840	12970	11831.90	12240.45	12491.99	93.98	140.78	12.91
samples500HTTP Request	500	0	0.00%	4914.38	2840	7353	6577.00	6769.35	7028.62	63.49	94.74	8.71
samples1000HTTP Request	1000	0	0.00%	9608.29	6268	12970	12146.20	12319.95	12519.00	62.65	94.03	8.61

## Incentive Microservice



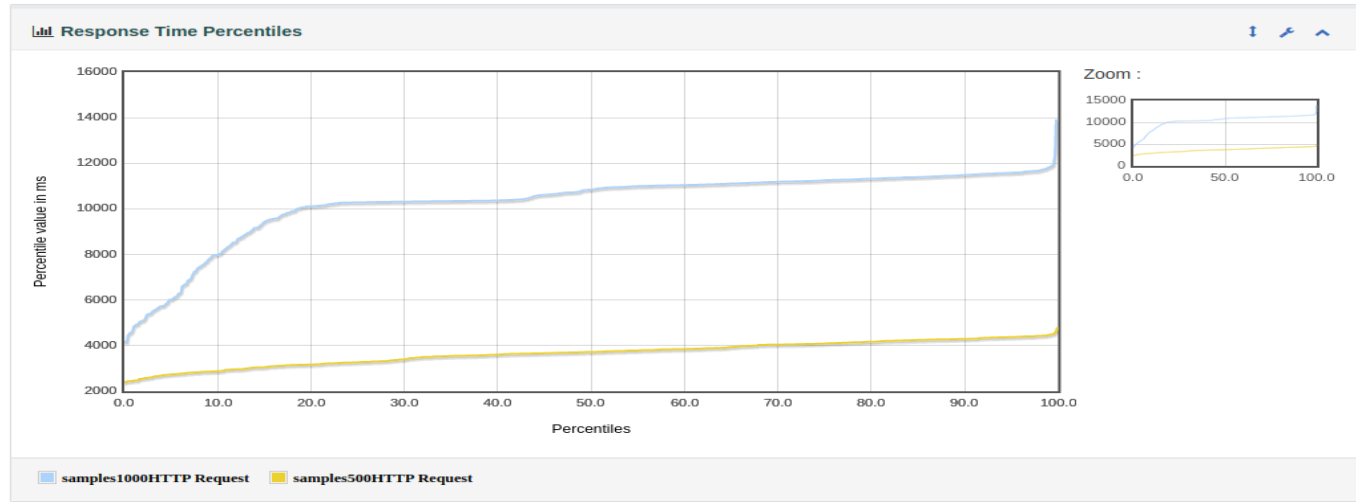
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	0	0.00%	5135.32	2861	7416	6158.50	6305.95	6747.91	147.67	44.49	27.28
samples500HTTP Request	500	0	0.00%	4608.70	2861	5745	5194.60	5248.95	5547.72	80.62	24.27	14.88
samples1000HTTP Request	1000	0	0.00%	5398.62	3657	7416	6257.90	6347.90	6830.90	99.48	29.98	18.39

Dealer Microservice



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	0	0.00%	3990.06	2209	6184	4785.50	5000.00	5198.91	180.35	47.17	26.18
samples500HTTP Request	500	0	0.00%	4403.81	2308	5326	5056.70	5144.85	5246.99	82.63	21.60	11.99
samples1000HTTP Request	1000	0	0.00%	3783.18	2209	6184	4519.40	4693.70	4974.89	120.24	31.45	17.46

Lead Microservice



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1500	2	0.13%	8093.03	2394	13915	11395.80	11541.00	11737.93	97.96	27.10	21.20
samples500HTTP Request	500	0	0.00%	3671.62	2394	4772	4295.80	4376.85	4481.84	92.37	25.49	19.96
samples1000HTTP Request	1000	2	0.20%	10303.73	4167	13915	11487.90	11584.90	11827.74	65.63	18.18	14.21

Microser	Microservice Name	Response Time(ms)	Response Time(ms)
----------	-------------------	-------------------	-------------------

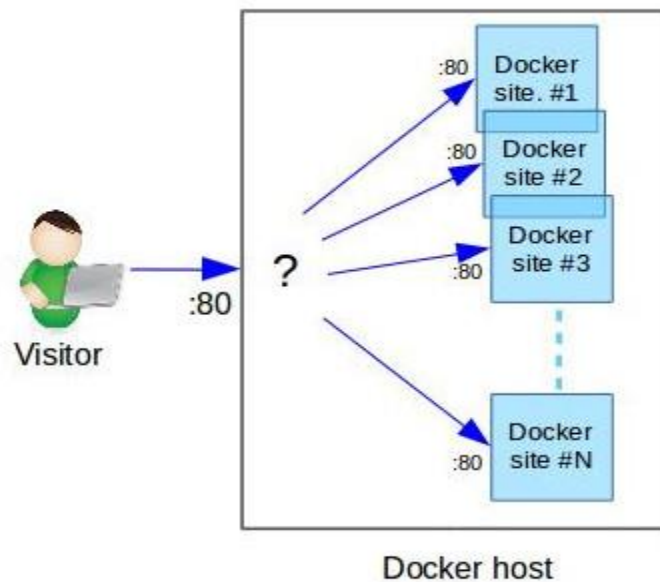


vice #		(500 samples)	(1000 Samples)
1	Part Service	4245.85	14590.98
2	Product Service	5381.52	6712.26
3	Compare Service	4818.51	14296.40
4	Incentives Service	3133.11	3481.54
5	Pricing Service	4914.38	9608.29
6	Dealer Service	4608.70	5398.62
7	Inventory Service	4403.81	3783.48
8	Lead Service	3671.6	10303.73

*Table 5.1 : Response time of the microservices for 500 and 1000 users*

From the above table we observe that the response time increases with increase in the number of requests.

## 5.5 Multiple Instances of Microservices



*Fig 5.4 : Architecture of multiple instances*

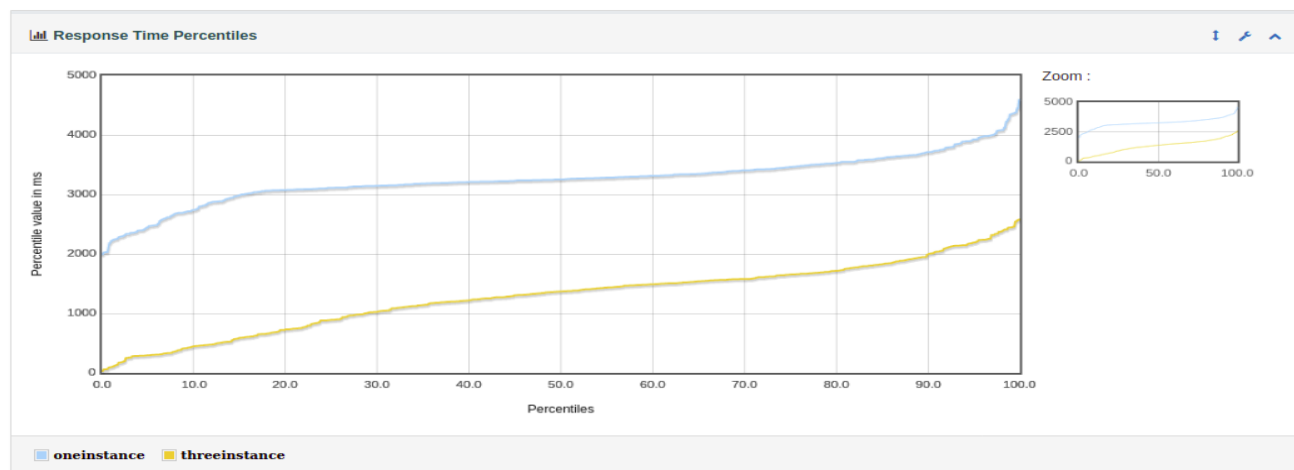
Multiple instances of microservices are deployed. Their response time is evaluated. Each instance of microservice receives requests on the basis of client side load balancing.

## 5.5.1 Performance under multiple instances

Response time of Microservice for 500 users (threads) is measured for three instances and is compared with the response time of a single instance

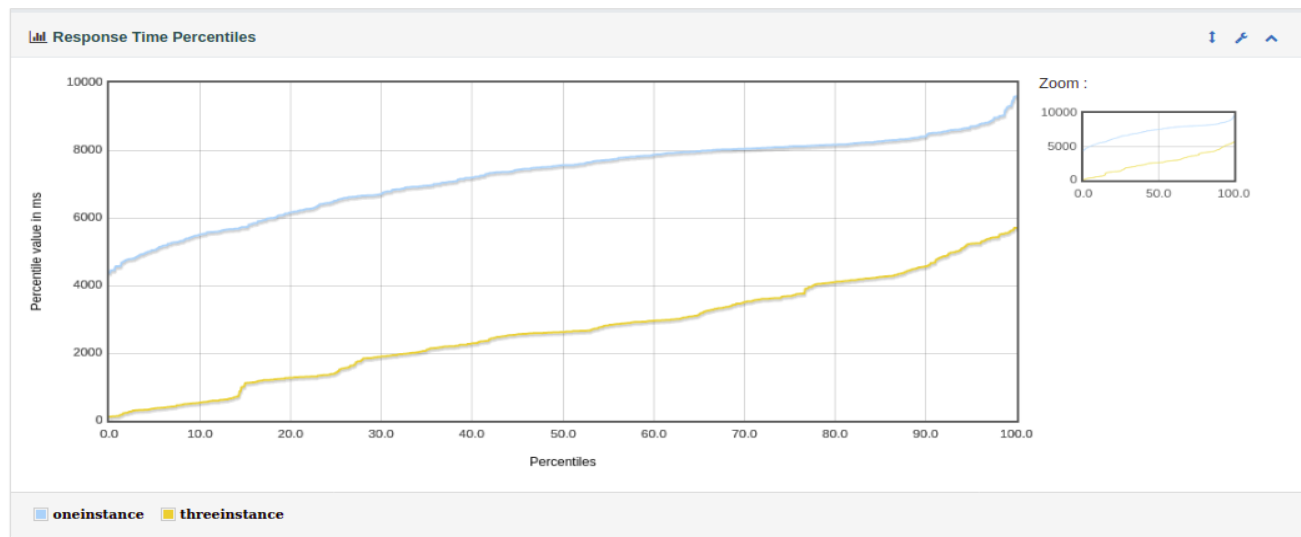
The following results are obtained

### Part Microservice



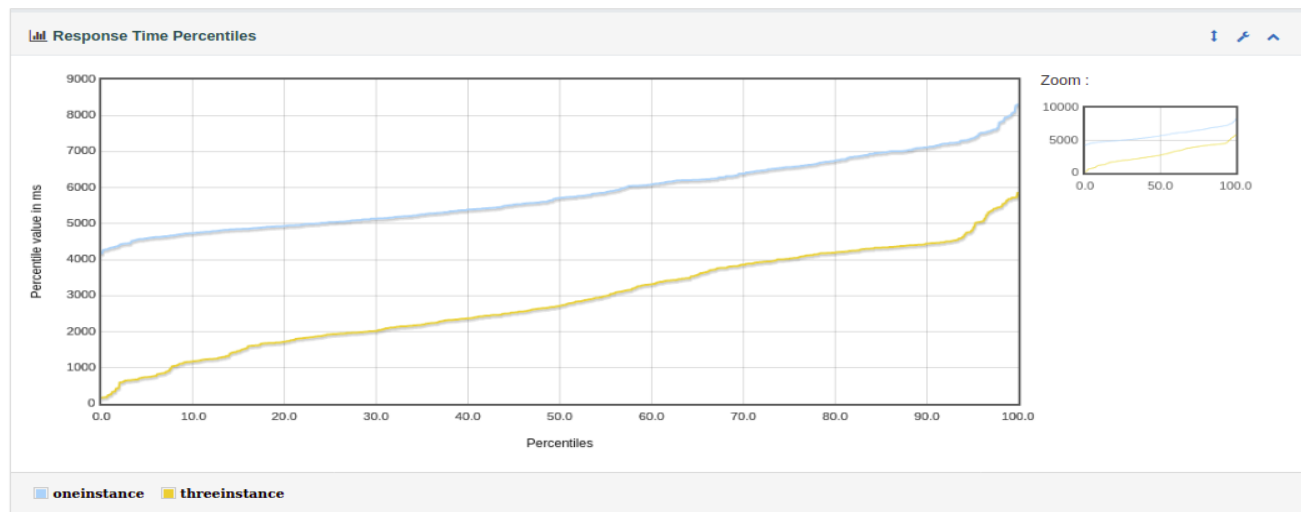
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	2273.02	30	4590	3531.90	3712.90	4083.97	163.75	65.07	30.09
oneinstance	500	0	0.00%	3260.84	1993	4590	3712.80	3925.90	4355.88	81.87	32.53	15.04
threeinstance	500	0	0.00%	1285.21	30	2579	2002.60	2194.85	2445.98	120.22	47.77	22.09

# Product Microservice



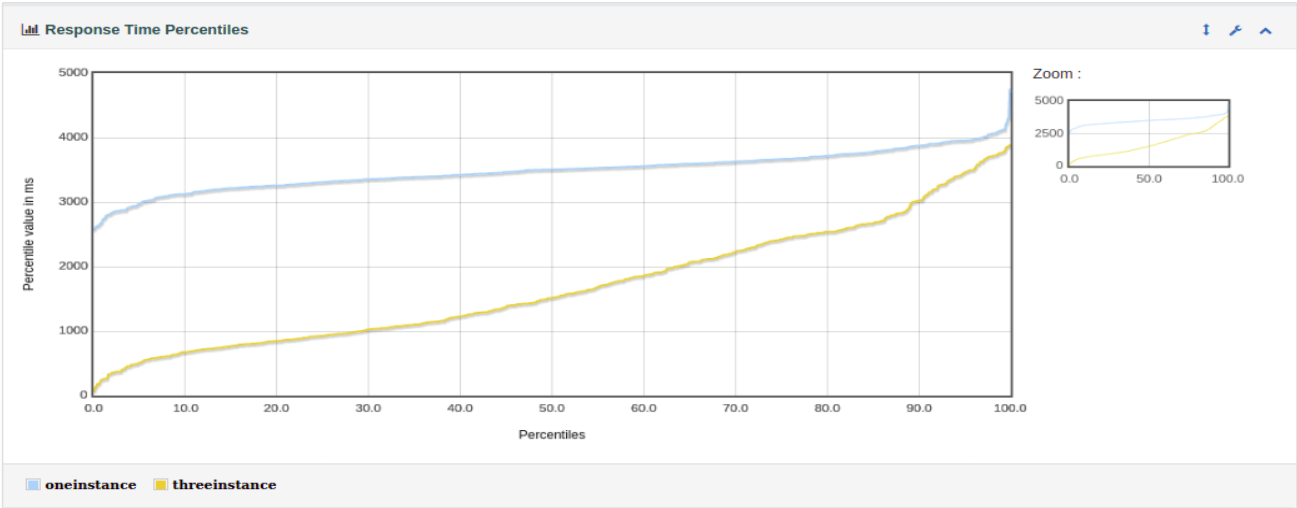
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	4955.04	112	9600	8166.70	8465.15	9014.58	89.21	2611.46	12.32
oneinstance	500	0	0.00%	7250.73	4373	9600	8462.30	8707.85	9304.67	44.61	1305.73	6.16
threeinstance	500	0	0.00%	2659.34	112	5700	4576.70	5241.35	5571.76	73.91	2163.48	10.21

# Compare Microservice



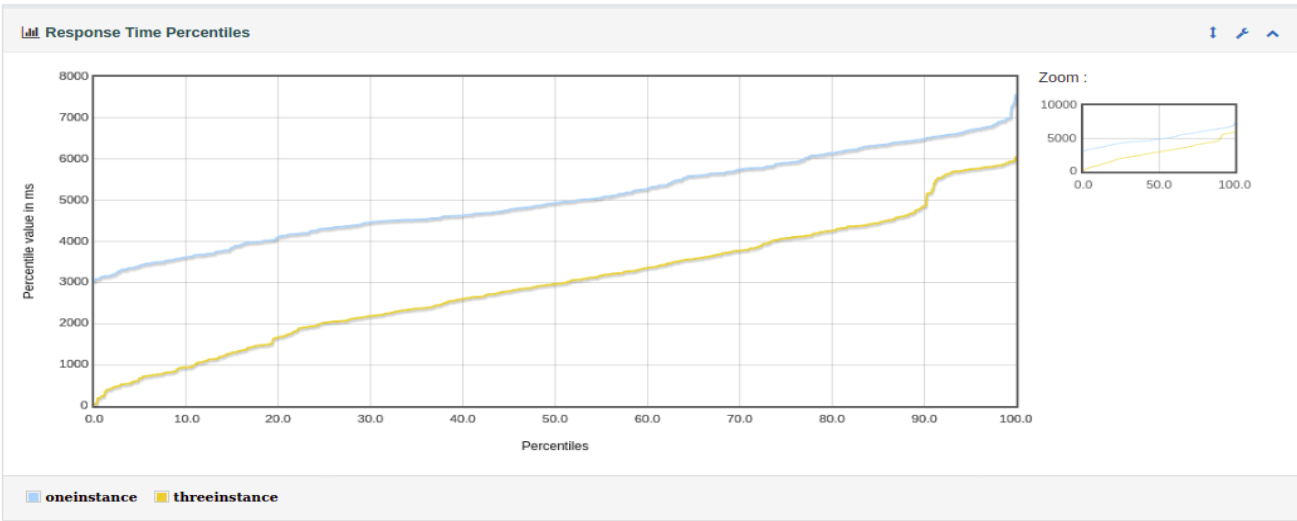
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	4357.70	167	8308	6754.80	7122.95	7833.75	104.21	66.78	27.30
oneinstance	500	0	0.00%	5835.07	4176	8308	7122.90	7378.45	8006.79	52.11	33.39	13.65
threeinstance	500	0	0.00%	2880.34	167	5858	4446.60	4901.45	5685.98	75.85	48.61	19.87

# Pricing Microservice



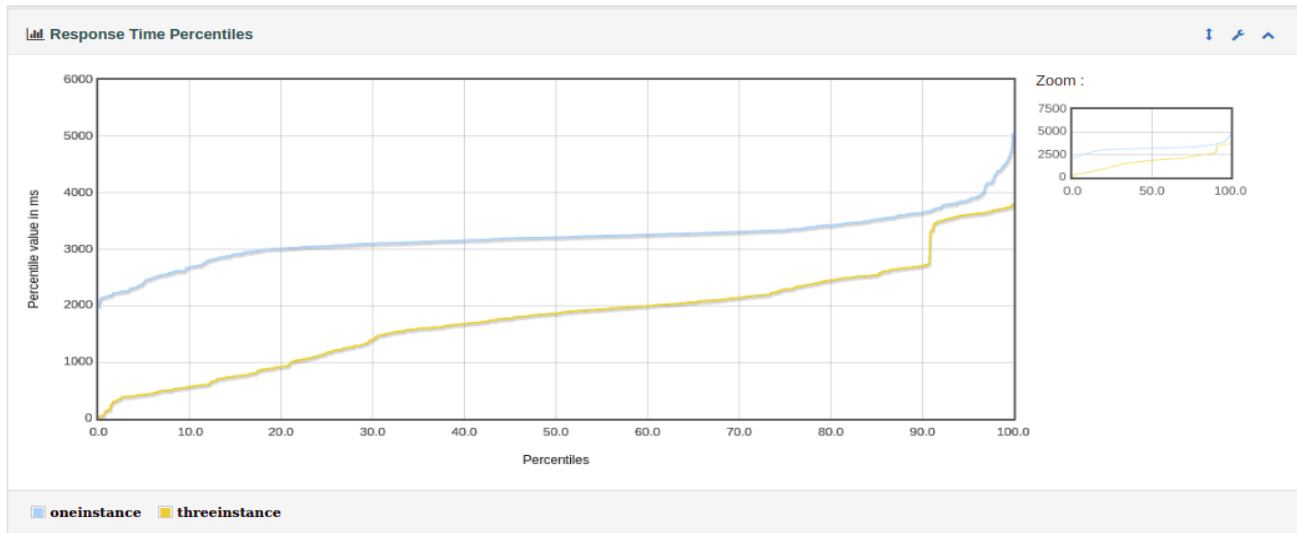
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	2593.06	77	4743	3740.80	3878.90	4067.92	188.64	72.18	35.22
oneinstance	500	0	0.00%	3489.36	2580	4743	3876.80	3956.95	4123.85	95.55	36.56	17.84
threeinstance	500	0	0.00%	1696.75	77	3883	3029.30	3469.25	3770.93	96.45	36.90	18.01

# Inventory Microservice



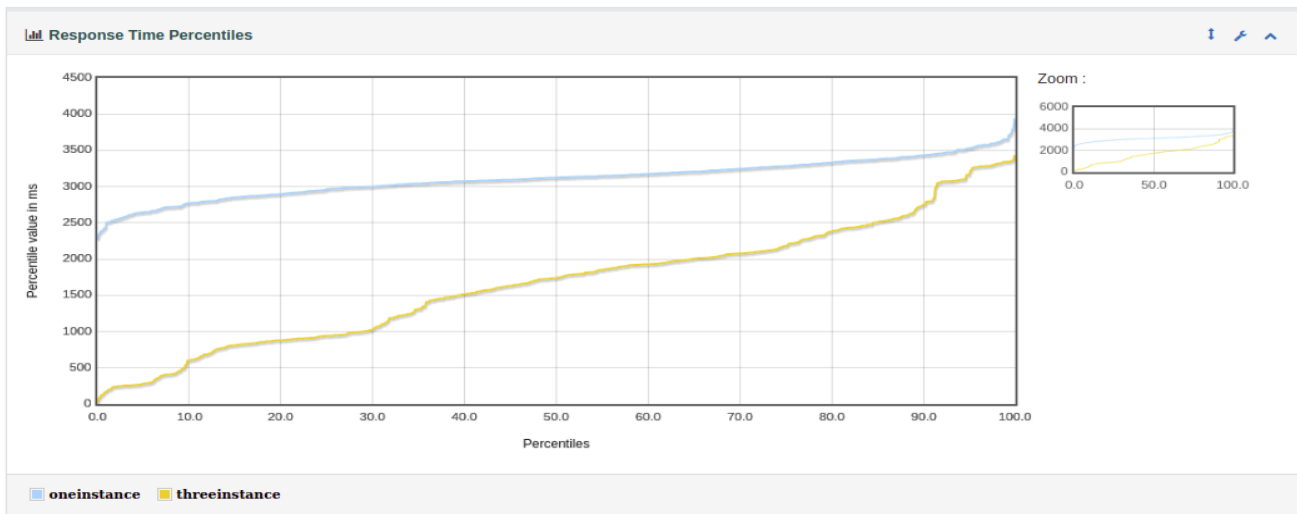
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	4022.12	43	7556	6148.90	6495.55	6906.68	122.35	182.57	16.79
oneinstance	500	0	0.00%	5045.18	3037	7556	6495.10	6704.75	6995.80	61.18	91.29	8.40
threeinstance	500	0	0.00%	2999.06	43	6048	4860.60	5757.90	5908.99	71.64	106.90	9.83

## Incentive Microservice



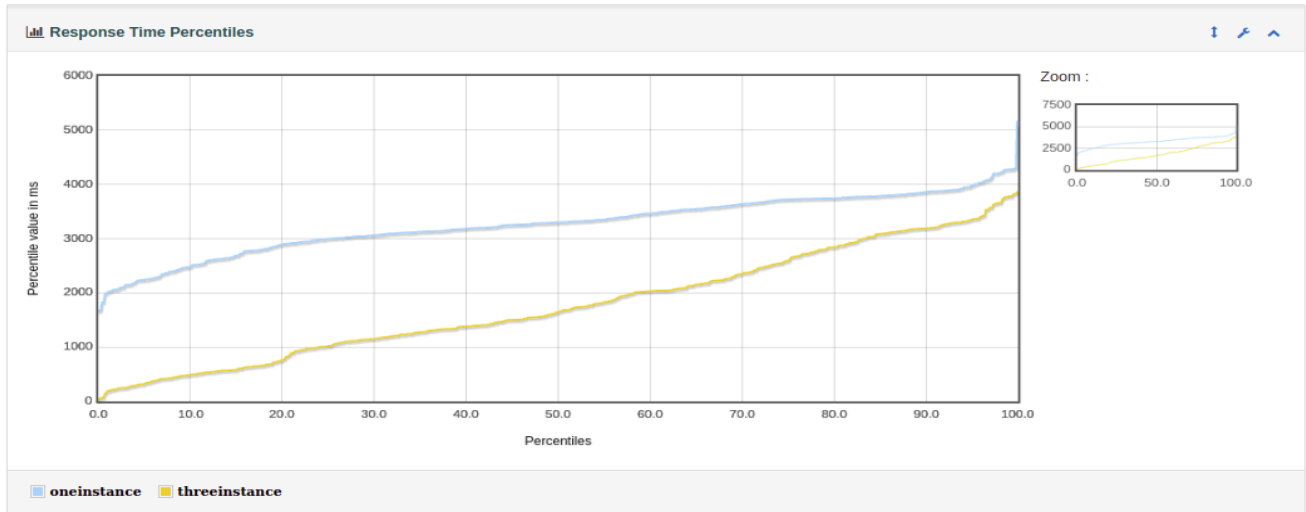
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	2507.14	29	5050	3565.80	3714.85	4355.56	181.09	54.51	33.41
oneinstance	500	0	0.00%	3208.39	1996	5050	3662.60	3872.00	4513.80	90.55	27.26	16.71
threeinstance	500	0	0.00%	1805.90	29	3795	2708.90	3614.90	3725.99	103.01	31.01	19.01

## Dealer Microservice



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	2391.96	34	3931	3342.90	3435.85	3612.93	186.85	48.85	27.11
oneinstance	500	0	0.00%	3113.12	2294	3931	3435.70	3532.80	3654.98	96.84	25.32	14.05
threeinstance	500	0	0.00%	1670.80	34	3424	2749.90	3220.20	3342.98	96.84	25.32	14.05

## Lead Microservice



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	2527.12	26	5160	3761.70	3860.95	4219.90	171.85	47.43	37.14
oneinstance	500	0	0.00%	3273.28	1667	5160	3860.70	3977.90	4274.00	85.93	23.71	18.57
threeinstance	500	0	0.00%	1780.96	26	3860	3190.90	3361.30	3780.96	93.18	25.72	20.14

Microservice #	Microservice Name	Response Time(ms) (One Instance)	Response Time(ms) (Three Instances)
1	Part Service	3260.84	1285.21
2	Product Service	7250.73	2659.34
3	Compare Service	5835.07	2880.34
4	Incentives Service	3489.36	1696.75
5	Pricing Service	5045.18	2999.06
6	Dealer Service	3208.39	1805.90
7	Inventory Service	3113.12	1670.80
8	Lead Service	3273.28	1780.96

Table 5.2 : Response time for single and three instances

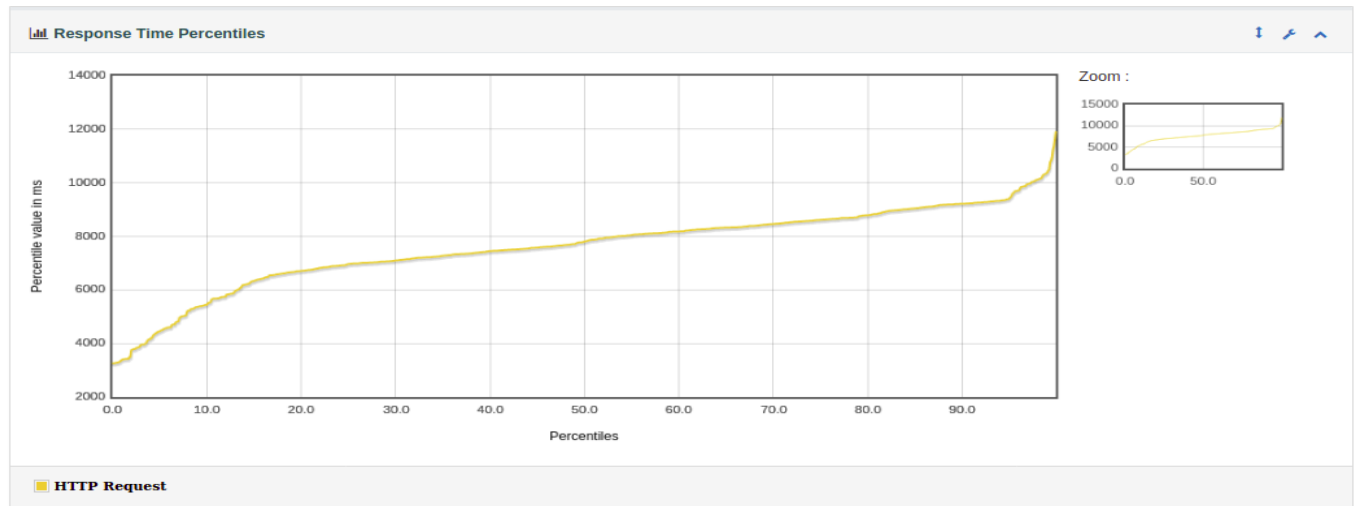
From the above table we observe that the performance can be increased by running multiple instances of each microservice.

## 5.6 Performance of Application in SOA

SOA application is built using SpringBoot framework and each service is tested for 1000 threads(users).SOA is a modular means of breaking up monolithic applications into smaller components, while Microservices provides a smaller, more fine-grained approach to accomplishing the same objective.Services in SOA have higher granularity compared to Microservices Architecture.Each microservice stores data independently while in SOA components share the same storage. For microservices, it's typical to use Cloud while for SOA Application Servers are much more common.

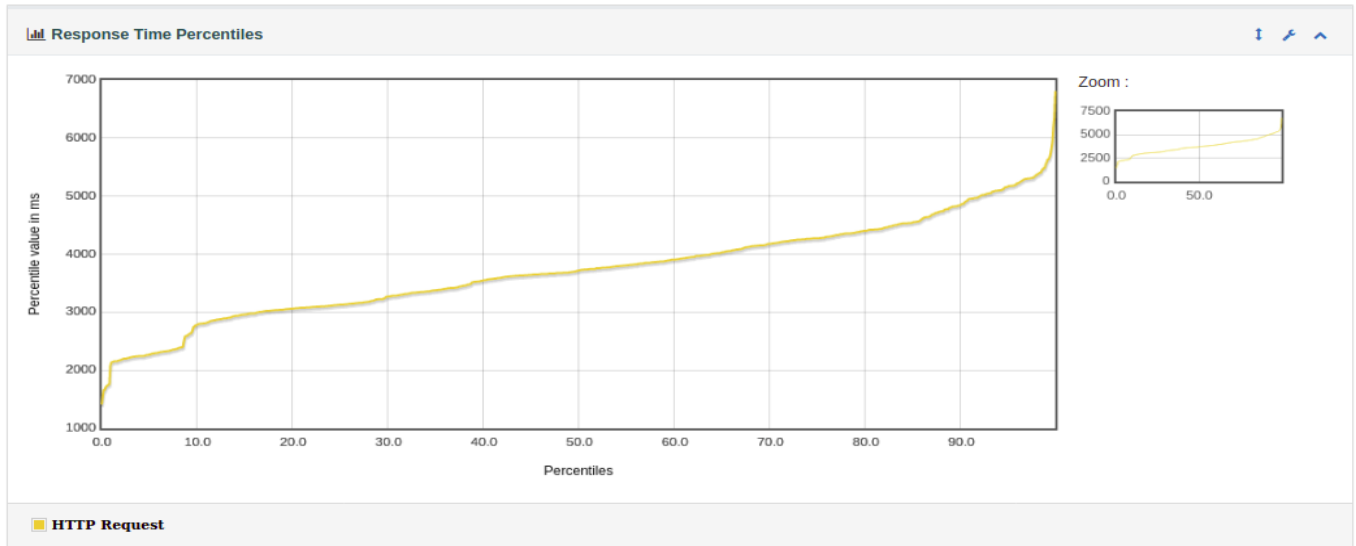
The following figures show the performance metrics of application in SOA

### PartProduct Service



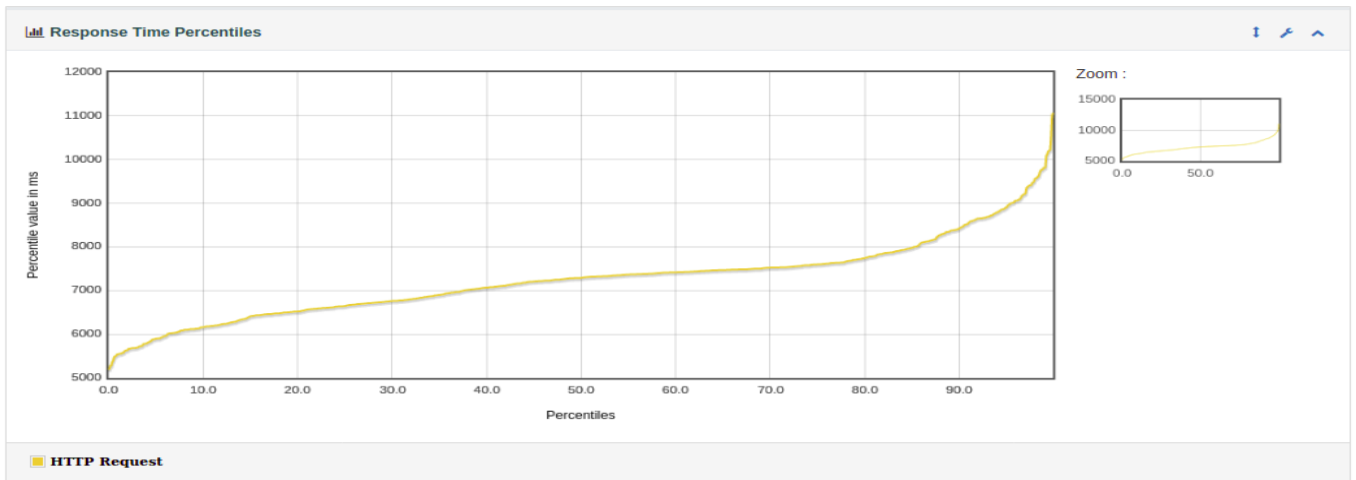
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	7616.00	3238	11929	9218.00	9426.30	10381.83	72.30	1072.49	11.63
HTTP Request	1000	0	0.00%	7616.00	3238	11929	9218.00	9426.30	10381.83	72.30	1072.49	11.63

## PricingIncentive Service



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	3733.55	1409	6811	4850.00	5171.55	5580.68	97.51	33.33	18.10
HTTP Request	1000	0	0.00%	3733.55	1409	6811	4850.00	5171.55	5580.68	97.51	33.33	18.10

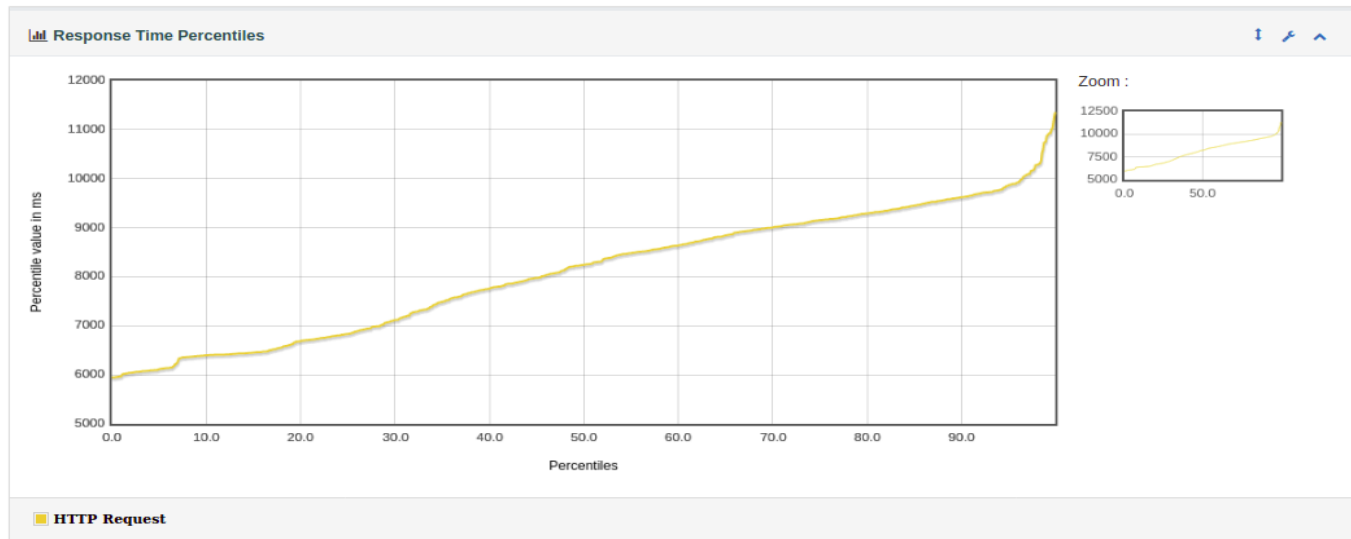
## DealerLead Service



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	7253.12	5191	11073	8428.10	8906.80	9812.99	79.88	21.47	14.43
HTTP Request	1000	0	0.00%	7253.12	5191	11073	8428.10	8906.80	9812.99	79.88	21.47	14.43



## CompareInventory Service

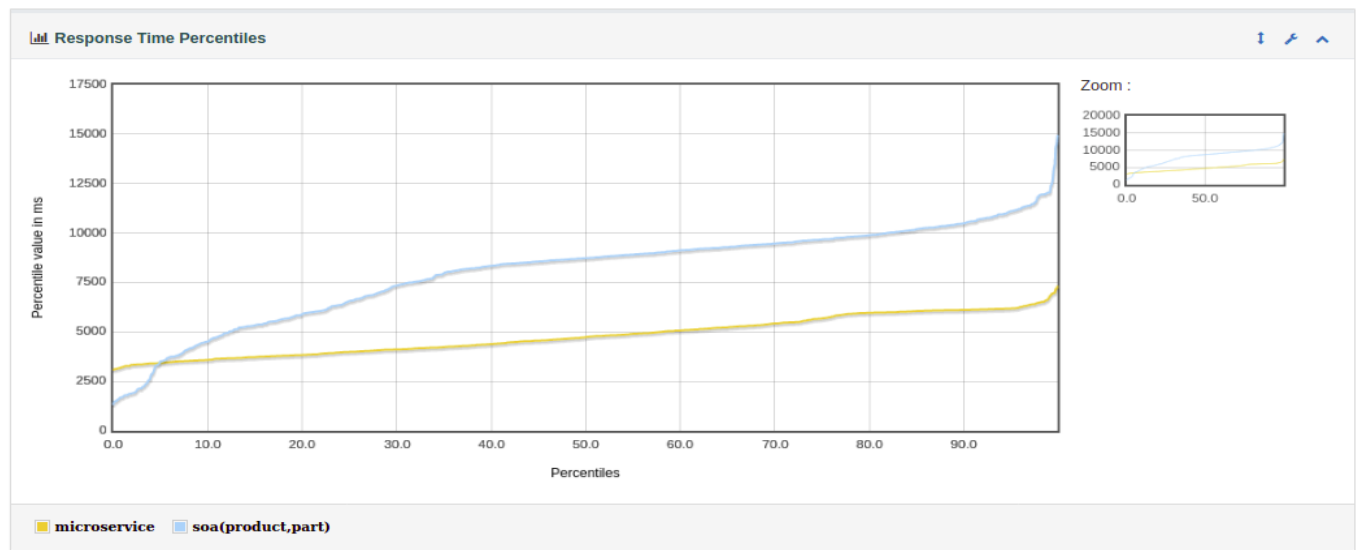


Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label ^	#Samples ^	KO ^	Error % ^	Average ^	Min ^	Max ^	90th pct ^	95th pct ^	99th pct ^	Transactions/s ^	Received ^	Sent ^
Total	1000	0	0.00%	8090.75	5943	11361	9621.00	9872.20	10870.84	68.05	72.58	13.58
HTTP Request	1000	0	0.00%	8090.75	5943	11361	9621.00	9872.20	10870.84	68.05	72.58	13.58

## 5.7 Comparison of Application Performance in SOA and Microservices

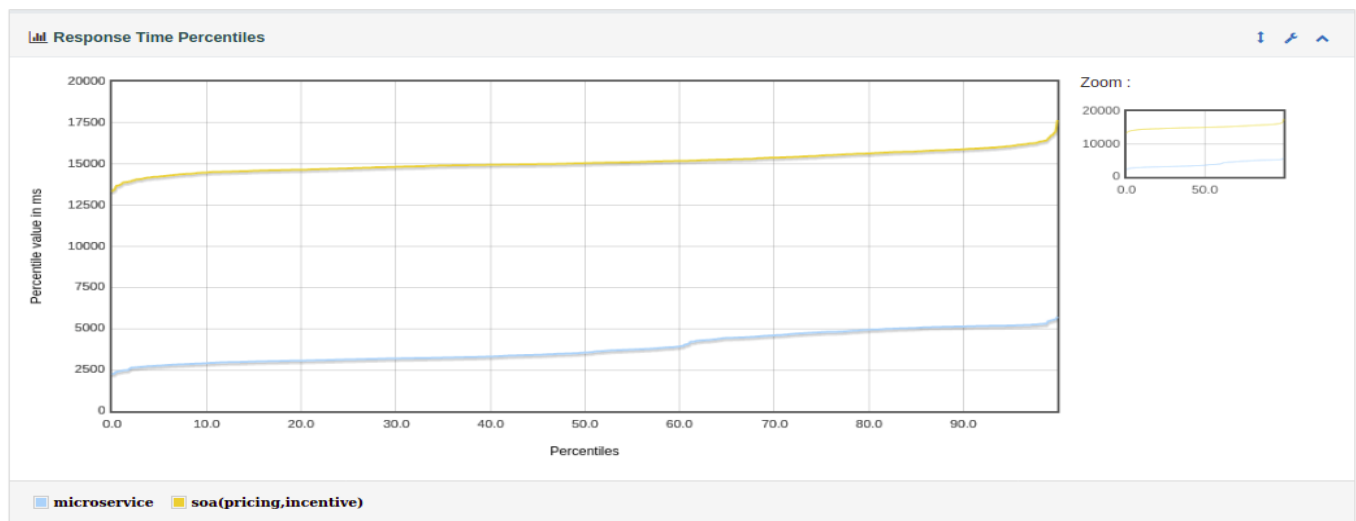
The response times of the Application in SOA and Microservices is compared. For each service in SOA, the average of the response times of the microservices which altogether have the same functionality as that of the service are compared. For example the response time of PartProduct Service is compared with the average of the response times of part microservice and product microservice.

## 5.7.1 PartProduct Service



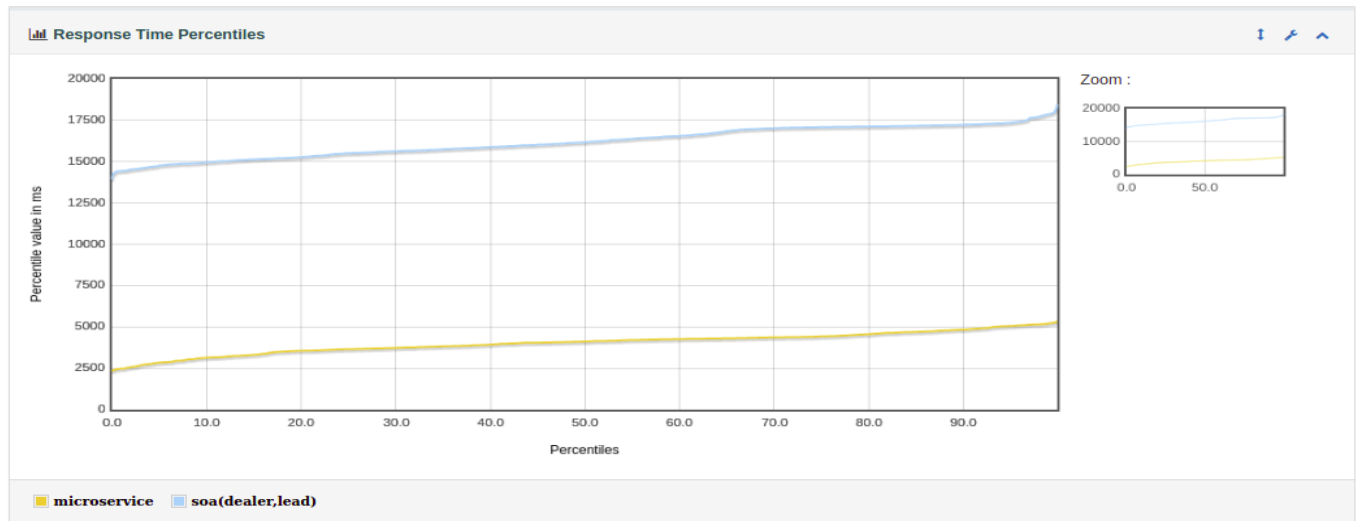
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	6447.13	1299	14959	9892.70	10495.20	11921.17	121.34	1800.10	19.53
microservice	1000	0	0.00%	4813.69	3049	7335	6108.90	6183.95	6721.41	124.25	1843.27	20.00
soa(product,part)	1000	0	0.00%	8080.58	1299	14959	10494.40	11104.80	12041.99	61.30	909.32	9.86

## 5.7.2 PricingIncentive Service



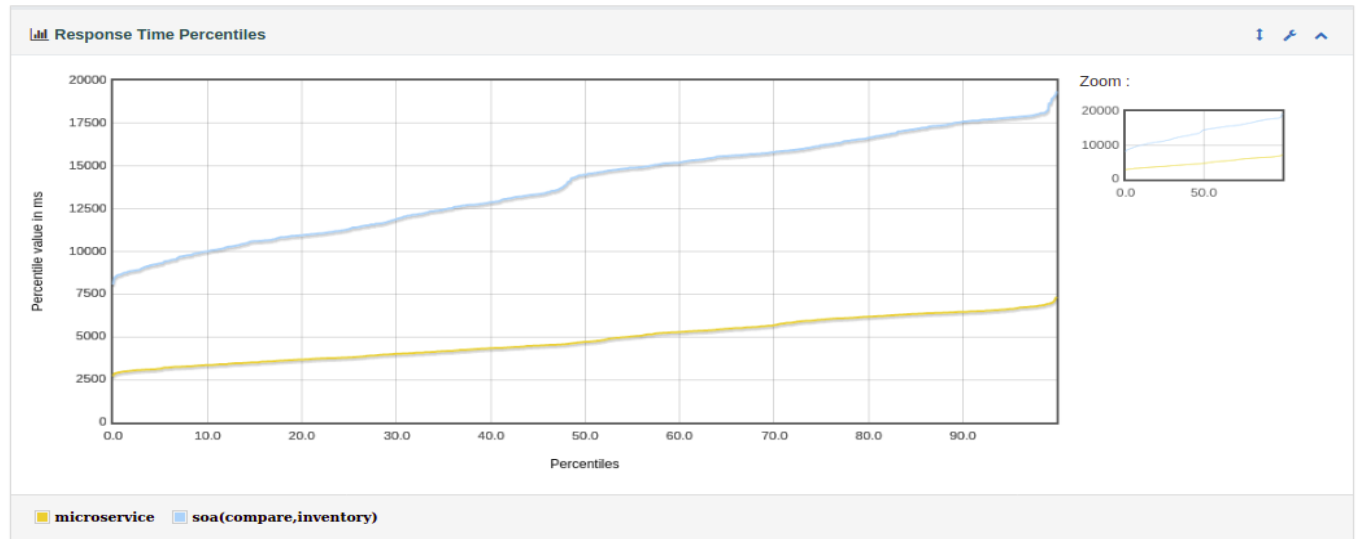
Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	9499.20	2220	17676	15641.90	15897.75	16353.74	106.10	36.27	19.69
microservice	1000	0	0.00%	3870.91	2220	5745	5136.90	5194.80	5459.95	161.24	55.11	29.93
soa(pricing,incentive)	1000	0	0.00%	15127.50	13315	17676	15897.50	16093.10	16589.33	53.57	18.31	9.94

## 5.7.3 DealerLead Service



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	10118.66	2308	18472	17126.90	17230.75	17722.85	99.65	26.78	18.00
microservice	1000	0	0.00%	4037.72	2308	5326	4839.70	5056.85	5218.80	154.87	41.61	27.97
soa(dealer,lead)	1000	0	0.00%	16199.60	13917	18472	17230.50	17355.55	17868.88	50.53	13.58	9.13

## 5.7.4 CompareInventory Service



Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	9376.96	2691	19378	16647.30	17607.80	18041.87	93.73	99.96	18.71
microservice	1000	0	0.00%	4866.44	2691	7353	6465.60	6636.00	6933.94	121.70	129.79	24.29
soa(compare,inventory)	1000	0	0.00%	13887.48	8066	19378	17606.60	17836.75	18271.81	46.86	49.98	9.35

<b>Service #</b>	<b>Service Name</b>	<b>Response Time(ms) (Microservice)</b>	<b>Response Time(ms) (SOA)</b>
<b>1</b>	PartProduct Service	4813.69	8080.58
<b>2</b>	PricingIncentive Service	3870.91	15127.50
<b>3</b>	DealerLead Service	4037.32	16199.60
<b>4</b>	CompareInventory Service	4866.44	13887.48

*Table 5.3 : Comparison of response times of application in Microservices and SOA*

From the above comparison we observe that the response time of the application is less in Microservice architecture than that of SOA. Hence we conclude from the above that Microservice architecture performs better than SOA.

## **CHAPTER 6**

### **CONCLUSION**

In this report, we have discussed SOA and Microservices architectures.

To compare both the architectures, software metrics related to coupling, response time and throughput are considered. We have implemented a Retail Vehicle application in microservices architecture using Spring boot. We have applied the identified metrics and analyzed the metric values.

We have implemented a Retail Vehicle application in SOA using Spring boot.

We have verified that the response time of the microservice application increases with increase in the number of users. Also the performance is improved when multiple instances of a microservice are run compared to a single instance.

We have verified that the application performs better in terms of response time and scalability when implemented in Microservices architecture compared to SOA.

In the future we would consider other parameters for comparison like reliability, error rate, memory usage for comparison of both the architectures.

## CHAPTER 7

### REFERENCES

- [1].Taibi, Davide, Valentina Lenarduzzi, and Claus Pahl. "Architectural patterns for microservices: a systematic mapping study." SCITEPRESS, 2018.
- [2].Raj, Vinay, and S. Ravichandra. "Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services." 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). IEEE, 2018.
- [3].Salah, Tasneem, et al. "Performance comparison between container-based and VM-based services." 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). IEEE, 2017.
- [4].Taibi, Davide, Valentina Lenarduzzi, and Claus Pahl. "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation." IEEE Cloud Computing 4.5 (2017): 22-32.
- [5].Qingqing, Zhang, and Li Xinke. "Complexity metrics for service-oriented systems." 2009 second international symposium on knowledge acquisition and modeling. Vol. 3. IEEE, 2009.
- [6].Lindvall, Mikael, Roseanne Tesoriero Tvedt, and Patricia Costa. "An empirically-based process for software architecture evaluation." Empirical Software Engineering 8.1 (2003): 83-108.
- [7].Richards, Mark. Microservices vs. service-oriented architecture. O'Reilly Media, 2015.
- [8].Alshuqayran, Nuha, Nour Ali, and Roger Evans. "A systematic mapping study in microservice architecture." 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, 2016.
- [9].Cerny, Tomas, Michael J. Donahoo, and Jiri Pechanec. "Disambiguation and comparison of soa, microservices and self-contained systems." Proceedings of the International Conference on Research in Adaptive and Convergent Systems. 2017.
- [10].Claus Pahl, and Pooyan Jamshidi. "Microservices: A Systematic Mapping Study." CLOSER (1). 2016.
- [11].Bhallamudi P, Tilley S, Sinha A. Migrating a Web-based application to a service-based system-an experience report. In 2009 11th IEEE International Symposium on Web Systems Evolution 2009 Sep 25 (pp. 71-74). IEEE.