

St. Cloud State University

## The Repository at St. Cloud State

---

Culminating Projects in Computer Science and  
Information Technology

Department of Computer Science and  
Information Technology

---

12-2022

### Application Development Using Microservice Architecture

Tonmoy Saha

Tonmoy Saha  
*St. Cloud State University*

Follow this and additional works at: [https://repository.stcloudstate.edu/csit\\_etds](https://repository.stcloudstate.edu/csit_etds)



Part of the [Computer Sciences Commons](#)

---

#### Recommended Citation

Saha, Tonmoy and Saha, Tonmoy, "Application Development Using Microservice Architecture" (2022).  
*Culminating Projects in Computer Science and Information Technology*. 41.  
[https://repository.stcloudstate.edu/csit\\_etds/41](https://repository.stcloudstate.edu/csit_etds/41)

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at The Repository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of The Repository at St. Cloud State. For more information, please contact [tdsteman@stcloudstate.edu](mailto:tdsteman@stcloudstate.edu).

# **Application Development Using Microservice Architecture**

By

Tonmoy Saha

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

December, 2022

Starred Paper Committee:  
Jie Hu Meichsner, Chairperson  
Omar Al-Azzam  
Ezzat Kirmani

## Abstract

Application development has always been a complex process. An application, once developed, also needs to be maintained and enhanced to add new requirements. Traditionally the application has been a monolithic entity. Different components in the application are tightly coupled and making a change has always been challenging. Microservice architecture breaks away from this monolithic approach and arranges the different functionalities as services. In a microservice architecture, individual services are developed to perform one function only.

This report demonstrates the application development process using the Microservice architecture. It explains the design, development, and deployment of a Microservice-based application. Market Place is an e-commerce application that consists of a collection of microservices working together to provide a buyer and seller platform to individuals. This application will allow sellers to showcase their products on this platform. The application consists of the following microservices: Product Microservice, Order Microservice, UI Microservice, and database Microservice. Similarly, the buyers can connect with the sellers directly in this application. This application is developed using the Spring Microservice framework, the services are hosted in Kubernetes. Docker is used for the containerization of services.

## Table of Contents

	Page
List of Tables .....	6
List of Figures .....	7
Chapter	
1. Introduction .....	9
2. Technology Overview .....	10
2.1 Spring Boot .....	10
2.2 HTML .....	11
2.3 CSS .....	12
2.4 JavaScript .....	12
2.5 Docker .....	12
2.6 Kubernetes .....	13
2.7 MySQL .....	15
3. Microservice Architecture and Design .....	16
3.1 Principles of Microservices .....	16
3.1.1 Single responsibility per service .....	16
3.1.2 Microservices are Autonomous .....	17
3.2 Microservices Benefits .....	18
3.2.1 Supports Polyglot Architecture .....	18
3.2.2 Elastically and Selectively Scalable .....	19
3.2.3 Allowing the co-existence of different versions .....	19

Chapter	Page
4. Demonstration of a Microservice application .....	20
4.1 Application Overview .....	20
4.2 Application Architecture .....	20
4.3 Microservice Implementation .....	21
4.3.1 Create a Docker image of the service .....	22
4.3.2 Using Kubernetes to Orchestrate Microservice .....	23
4.4 Application Use Case .....	28
4.4.1 Use Case - General User Login .....	30
4.4.2 Use Case - General User purchase .....	31
4.4.3 Use Case - General User Register .....	32
4.4.4 Use Case - General User Update Profile .....	33
4.4.5 Use Case - General User Purchase History .....	33
4.4.6 Use Case - Admin User Login .....	34
4.4.7 Use Case - Admin User Find Product .....	35
4.4.8 Use Case - Admin User Add Product .....	36
4.4.9 Use Case - Admin User Modify Product .....	36
4.5 Application Screenshots .....	37
4.5.1 Landing/Main Page .....	37
4.5.2 The Login Page .....	38
4.5.3 Admin Home .....	38
4.5.4 Add Product .....	39
4.5.5 Find product .....	39

Chapter	Page
4.5.6 Modify Product .....	39
4.5.7 User Home .....	40
4.5.8 User Profile Update .....	40
4.5.9 Purchase History .....	40
5. Demonstrating the advantages of Microservice .....	41
5.1 Ease of Deployment and Maintenance .....	41
5.2 High Elasticity and Selectively Scalable .....	41
5.3 Allowing the Co-existence of Different Versions .....	42
5.4 Better Quality of Service .....	42
5.5 Co-development and speed of delivery .....	43
6. Conclusion.....	44
References .....	45

**List of Tables**

Table	Page
1. General User Login Use Case Description .....	30
2. General User purchases Use case Description .....	31
3. General User Register Use Case Description .....	32
4. General User Update Profile Use Case Description .....	33
5. General User purchases History Use case Description .....	33
6. Admin User Login Use Case Description .....	34
7. Admin User Find Product Use Case Description .....	35
8. Admin User Add Product Use Case Description .....	36
9. Admin User Modify Product Use Case Description .....	36

## List of Figures

Figure	Page
1. The basic block diagram of Spring Framework .....	11
2 Overview of Docker Architecture .....	13
3. Kubernetes Basic Working .....	14
4. Block Diagram depicting Various Kubernetes components.....	15
5. Block depiction of Microservice Vs Monolithic Design .....	16
6. Diagram depicting Various components in a Microservice.....	17
7. Diagram showing the polyglot architecture .....	18
8. Showing the CO-existence of different versions of the service .....	19
9. Various Microservice components in the application .....	21
10. How to create Docker image .....	22
11. All the images that have been created in the project .....	23
12. YAML file for UI service of POD definition .....	23
13. YAML file for Product service of POD definition .....	24
14. YAML file for Order service of POD definition .....	24
15. YAML file for Database service of POD definition .....	25
16. YAML file for UI service of Service Definition .....	25
17. YAML file for Product service of Service Definition .....	26
18. YAML file for Order service of Service Definition .....	26
19. YAML file for Database service of Service Definition .....	27



Figure	Page
20. Shows all the PODs that are created in the system .....	27
21. Shows all the Services that are created in the system .....	28
22. Interaction Diagram for Market Place Application .....	28
23. Use case Diagram .....	29
24. Landing Screen of The Market Place App .....	37
25. Login Screen of The Market Place App .....	38
26. Admin Home Screen of The Market Place App .....	38
27. Add Product Screen of The Market Place App .....	39
28. Find Product Screen of The Market Place App .....	39
29. Modify Product Screen of The Market Place App .....	39
30. User Home Screen of The Market Place App .....	40
31. User Profile Update Screen of The Market Place App .....	40
32. Purchase History Screen of The Market Place App .....	40

## Chapter 1: Introduction

Microservices are an architectural style in which applications are developed as physically separated modules. The microservice-based application provides a simple yet powerful design, ease of development, flexible deployment cycle, speed of agility, and scalability [2]. Microservices started from the idea of Hexagonal Architecture. It is also known as the Ports and Adapters patterns [3]. Two fundamental principles of Microservices are single responsibility and autonomy. The single responsibility principle states that a unit should only have one responsibility. If any unit has more than one responsibility, it becomes tightly coupled. Autonomous means microservices are independently deployable and can perform a business capability independently.

Within a microservice, it could follow any of the existing patterns. The most popular one is the REST web service. Within the service, the code is stacked up in layer architecture. The controller layer uses the REST protocol to communicate with the caller. Below the Controller layer is the Service layer which communicates with the Domain layer. In this layer, the data are fetched or modified and persisted in the storage system, typically a Database.

In this project, a working application is developed using Microservice Architecture. It is called the Market Place application. The Administrative user act as a moderator of the application. The second kind of user is the Buyer and seller of products, goods, and services. Product information will be stored in the Database and displayed to the seller via a service call. Each transaction is persisted in the database for future reference. This application can add new business functionality without interfering with the existing running application. In the same way, an existing functionality could also be removed without hampering the rest of the application.

## Chapter 2: Technology Overview

This Chapter outlines the technologies that have been used in this project work. The microservice architecture in this project has been built around the following technologies: Spring Boot framework, Java programming language, HTML, CSS, and JavaScript. For Containerization, Docker is used, and for Orchestration, Kubernetes is used. MySQL has been selected as the database storage.

The Main technologies used in this paper are described below:

### 2.1 Spring Boot

It is a framework for building Java applications; Spring is one of the most popular Java frameworks. It gained popularity for developing Monolithic applications, primarily with its MVC architectural pattern. However, with the arrival of microservices, Spring has bought a new slim, convention-based framework that would cater to the demands of microservices. Spring Boot is the latest offering from Spring. Developers can build a standalone microservice with Spring Boot's help without Spring Framework's complexities. Figure 1 shows the Architecture diagram for the Spring Framework.

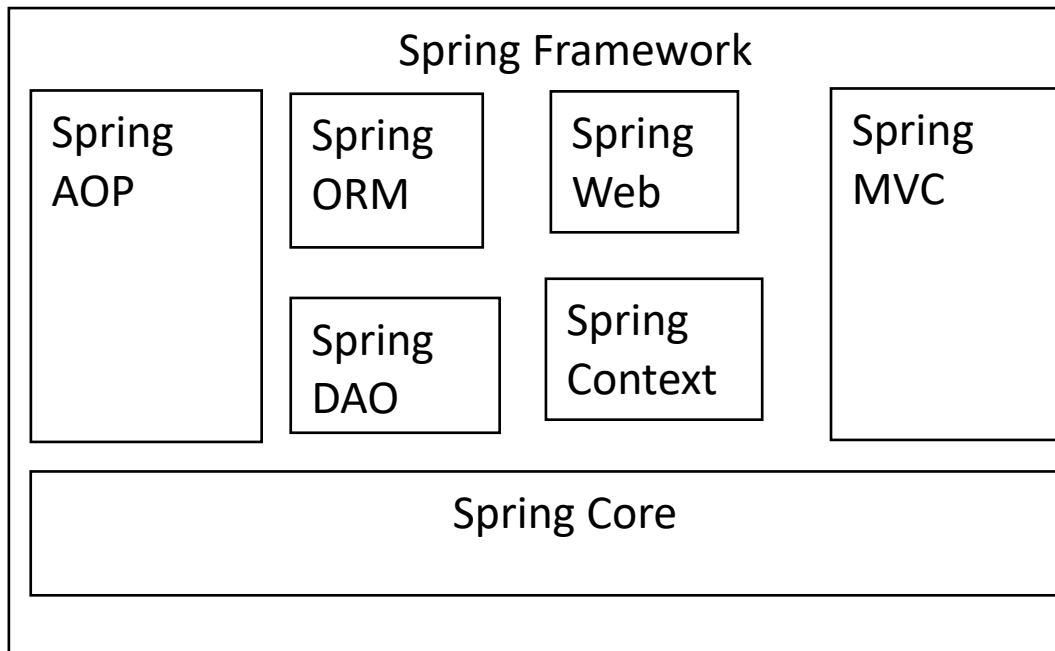


Fig. 1. Basic block diagram of Spring Framework

Some of the advantages of Spring Boot [1] are listed below:

- Lightweight
- Minimum Configuration
- Built-in Tomcat
- Ideal for microservice stand-alone architecture

## 2.2 HTML

HTML means Hyper Text Markup Language [9], which is used for creating web pages. HTML describes and defines the structure of web pages. It builds the pages with a series of Elements. These elements direct the browser what would be the content of the page. It creates a static display of the web page. All the components of a web page that are visible to the user are provided by HTML structure. To make it dynamic java scripts are applied to the HTML components.

### **2.3 CSS**

CSS stands for Cascading Style Sheets [9]. It affects the presentation of the web page. HTML elements could be rendered or presented in the browser in various ways; CSS tells how this could be done. CSS is not a programming language but a markup language.

### **2.4 JavaScript**

JavaScript is a scripting language or just in time-compiled programming language [9]. It is responsible for making the static HTML content of a web page dynamic. The standards for JavaScript are the ECMAScript Language Specification (ECMA-262) and the ECMAScript Internationalization API specification (ECMA-402) [8].

### **2.5 Docker**

Docker is a containerization technology [5]. It is an open-platform technology. It is used in developing, packaging, and deploying an application. Docker helps in separating the application from the infrastructure. This separation facilitates delivering the software in a faster manner. Docker provides an abstraction or isolation layer for the software to run. The container works independently of the host hardware or software configuration. Isolation means the containers run in a different environment from the host and each other. Figure 2 shows the architecture of Docker. Figure 2 is adapted from 'Docker Overview' [7].

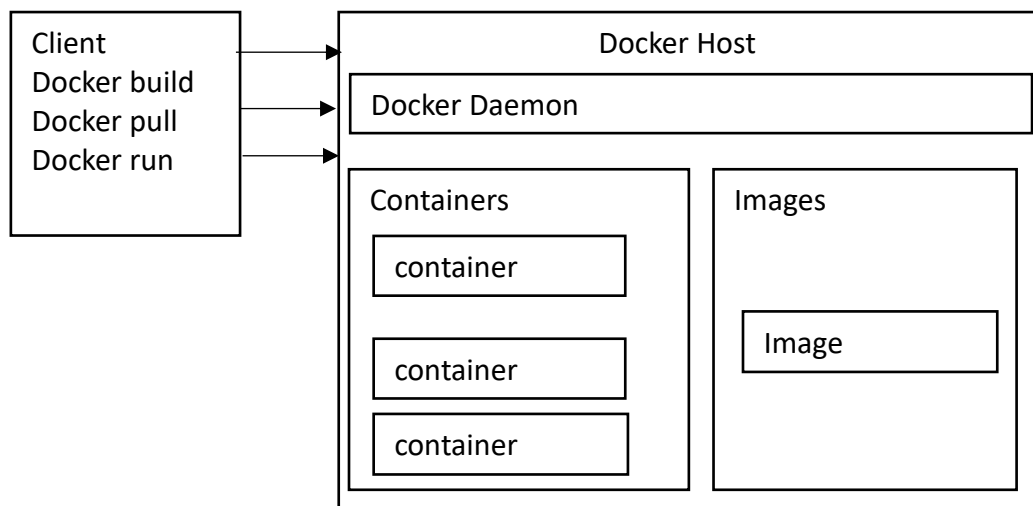


Fig. 2. Overview of Docker Architecture

## 2.6 Kubernetes

Kubernetes is an open-source container orchestration engine. It is used for automating deployment, scaling up applications, and monitoring and managing the application. The Cloud Native Computing Foundation hosts it. Kubernetes plays a significant part in microservice-based architecture in deploying services as an independent unit. Kubernetes Basic Modules are shown in Figure 3. This figure is taken from “Learn Kubernetes” [6].

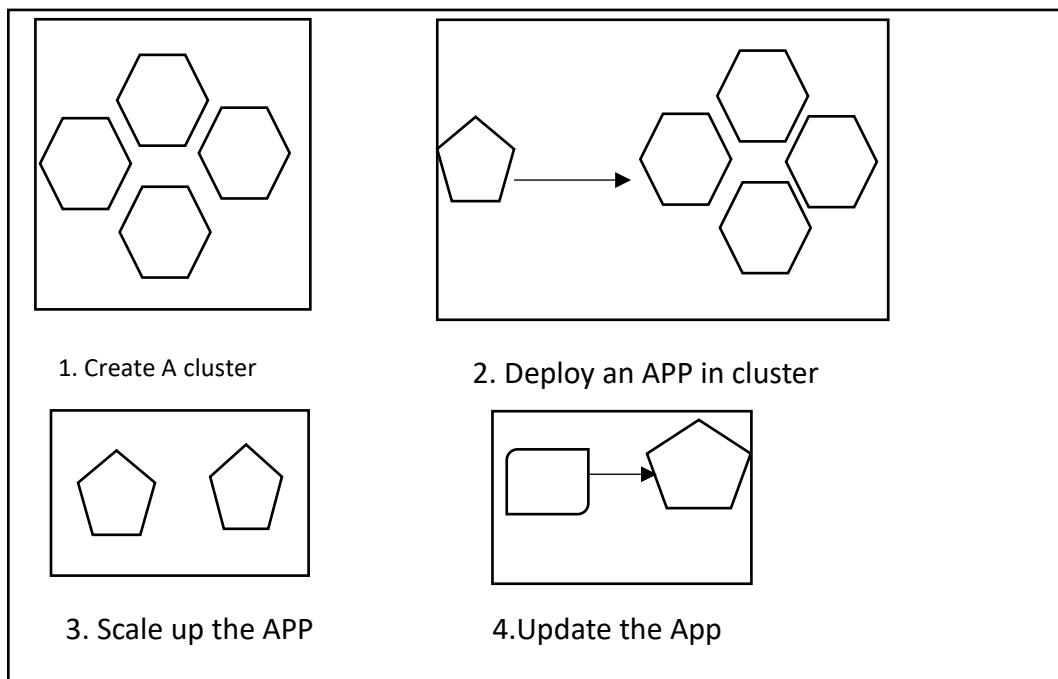


Fig. 3. Kubernetes Basic Working.

Kubernetes cluster consists of a Master node and a collection of worker nodes. Kubernetes communicates with the nodes with the help of the API server. It has built-in storage in 'etcd'. The Scheduler and controller are used for managing the nodes. In Kubernetes Pod is the basic unit of the Kubernetes platform. The pod is nothing but Docker containers. A Pod can have more than one container and share namespace, resources, and security features. A basic block-level component diagram is shown in Figure 4. This Figure is adapted from Docker and Kubernetes for Java Developers [4].

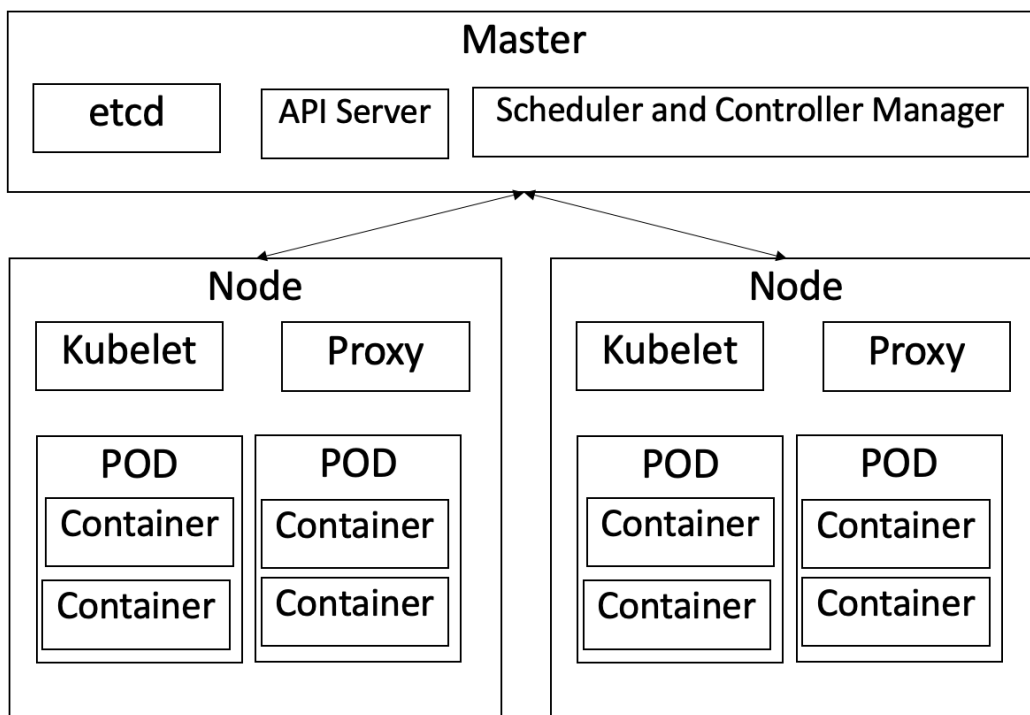


Fig. 4. Block Diagram depicting Various Kubernetes component

Kubernetes Service is an abstraction through which one or more POD can be accessed through networks. Every service in Kubernetes has its own IP address and port number.

They have the following features [4]:

- Service is permanent
- Service offers built-in load balancing
- It has a permanent IP address and port number
- They connect to a group of Pods and expose them to outside traffic

## 2.7 MySQL

MySQL is an open-source system, SQL stands for Structured Query Language. It is a relational database.



### Chapter 3: Microservice Architecture and Design

Microservices are an architectural style, and it was not invented but rather evolved from the existing architectural design. It is one of the most popular architectural design patterns today [2]. Following this design, one can develop a very agile, fast, and scalable solution. Microservice provides an option to develop physically separated modular applications. Microservices trace back their origin from the Hexagonal Architecture, which is also known as Ports and Adapters patterns.[3]

#### 3.1 Principles of Microservices

Single responsibility and Autonomous are the two key principles of Microservices [2].

##### 3.1.1 Single responsibility per service

One of the principles of the SOLID design pattern is Single responsibility. It means a single unit should have only one responsibility. A service should have only one responsibility of work. If it performs more than one responsibility, then tight coupling occurs.

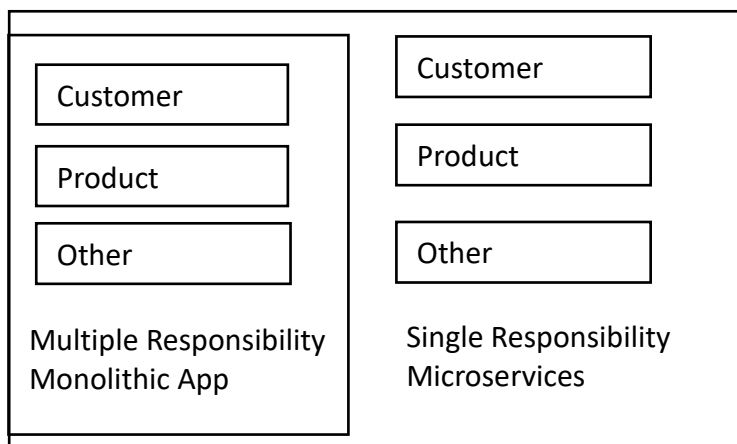


Fig. 5. Block depiction of Microservice Vs Monolithic

Figure 5 has been adapted from “Spring 5.0 Microservices” [2]. As depicted in Figure 5, Customer, Product, and Order are three different features of an e-commerce application. If all three are built in a single application, it will result in a tightly coupled monolithic application. In the microservice world, each feature will be a separate service, and changing one service will not impact the other one. Each service here is responsible to perform only one business task.

### 3.1.2 Microservices are Autonomous

Each Microservices are physically separate application or service. They could be independently built, compiled, and deployed. It includes all the dependencies like the libraries and execution environment.

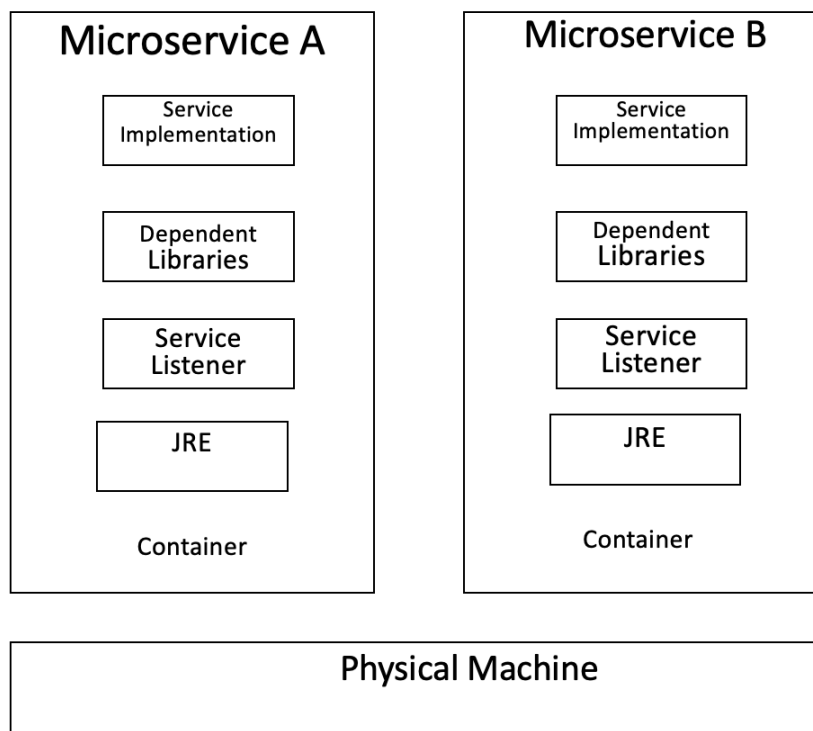


Fig. 6. Diagram depicting Various components in a Microservice

As shown in Figure 6, microservices get their own containers Container technology like Docker is ideal for this purpose.

### 3.2 Microservices Benefits

There are various benefits of Microservices over monolithic applications. A few major benefits [2] are discussed below:

#### 3.2.1 Supports Polyglot Architecture

As microservices are separate services and can be deployed and tested individually. They can be of different technology or different versions of the same technology. Figure 7 has been taken from “Spring 5.0 Microservices” [2].

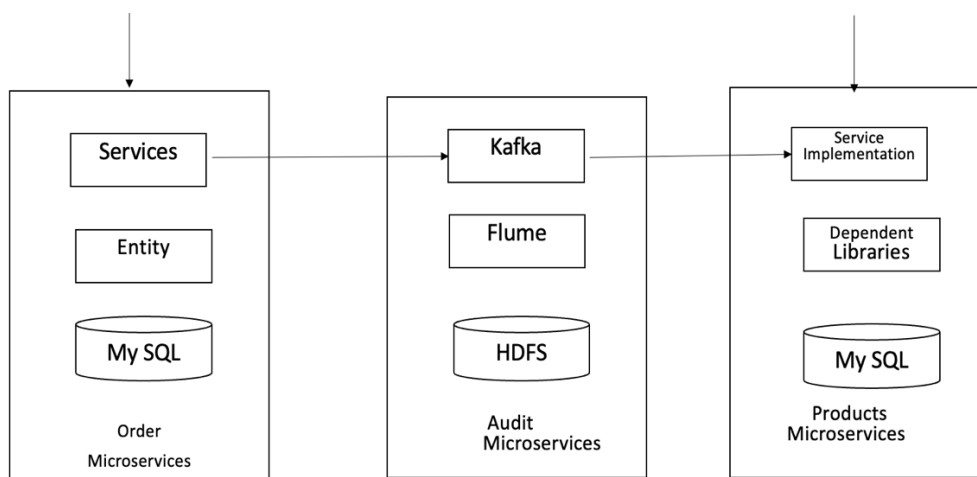


Fig. 7. Diagram showing the polyglot architecture

The Order microservice and Product microservice stores data in a relational database, whereas the Audit microservice stores data in Hadoop File System. Moreover, the Order Microservice could be in Java technology and the Products microservice could be developed in .Net technology.

### 3.2.2 Elastically and Selectively Scalable

In a monolithic application, there is no option to scale a particular feature or functionality as it is packaged as a single war or ear. A lot of resources go unutilized. That is not the case in microservice, as each microservice is an independent entity that could be scaled up or down independently. As scaling could be selectively applied the cost of resource are less and better utilized.

### 3.2.3 Allowing the co-existence of different versions

In a microservice architecture, one can have different versions of the same functionality in the production environment.

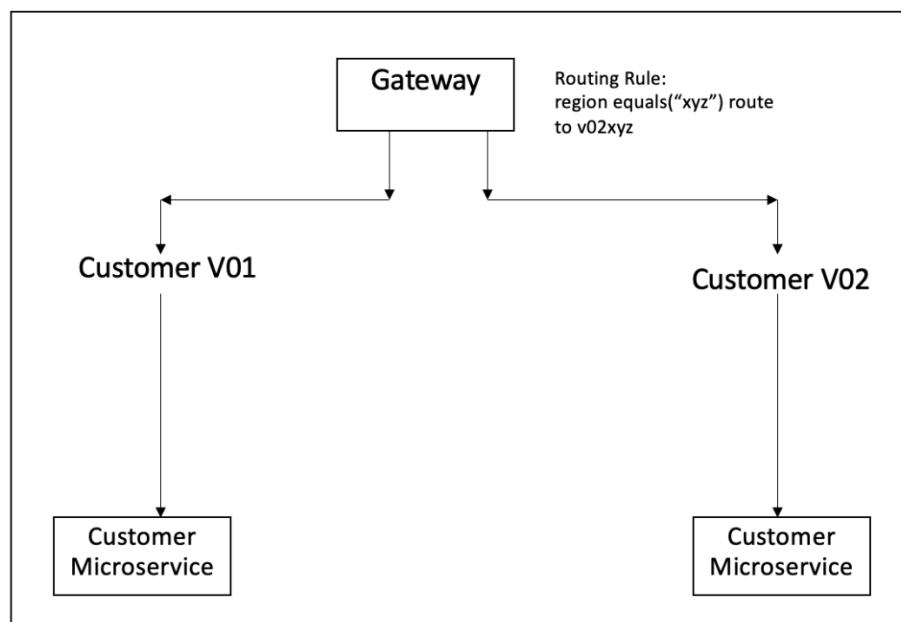


Fig. 8. Showing the CO-existence of different versions of the service

As shown in Figure 8, let us assume we need to have a new business requirement to be rolled out to customers in a particular region. This could be very easily achieved by having two versions of the same feature in production as V01 and V02. Then we could apply a routing rule at the Gateway level to route the traffic to the appropriate service based on the region.

## **Chapter 4: Demonstration of a Microservice application**

This Chapter describes the different features and components of a Microservice-based application. The first section gives an overview of the different functionalities of the application. Then there is a description of the system architecture. Then there are use case diagrams, flow charts, and screenshots of the application features.

### **4.1 Application Overview**

The Market Place application is a simple e-commerce application. Where a user/customer can buy different products that are available in the application. This application has two types of users Admin users and general users. Admin users can ADD or Modify new products in the system. Where the general users can create a login, manage the user profile, can purchase items from the e-commerce site, also can view a list of existing purchases.

### **4.2 Application Architecture**

This e-commerce application Market Place has been designed as a Microservice Architecture. Here the components are loosely coupled, and each functionality could be deployed and maintained separately. To demonstrate the microservice concept, the application has been divided into four individual microservice: Market Palace APP, the product microservice, the order microservice, and the database microservice. A basic block diagram in Figure 9, shows the microservices.

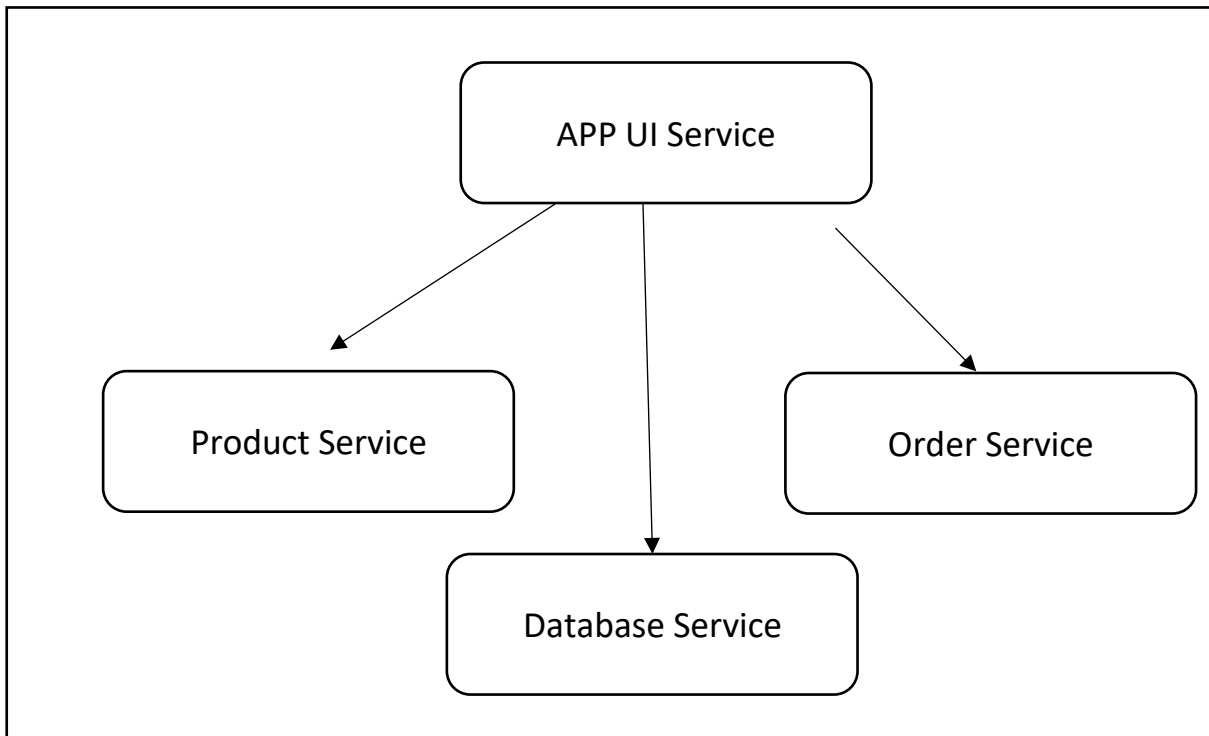


Fig. 9. Showing the various Microservice component in the application

### 4.3 Microservice Implementation

Each microservice is programmed in the spring boot framework of Java. The UI service has implemented Spring security to enable the login feature. The product service and order service are Spring boot RESTFUL APIs. Database service is out of the box MySQL image. Docker and Orchestration engines like Kubernetes have played a big role in popularizing the microservice design. In this project, both have been used. Docker for containerization and Kubernetes for Orchestration. A cut-down version of Kubernetes called mini Kube has been used to simulate the Kubernetes functionality.

The following steps have been followed to containerize the services:

### 4.3.1 Create a Docker image of the service

In the root folder of the service, create a Docker file. Below is the Docker File used in the UI microservice.

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} prj-mkt-place-login.jar
ENTRYPOINT ["java", "-jar", "/prj-mkt-place-login.jar"]
EXPOSE 8085 443
```

Log in to the Docker hub using the below command

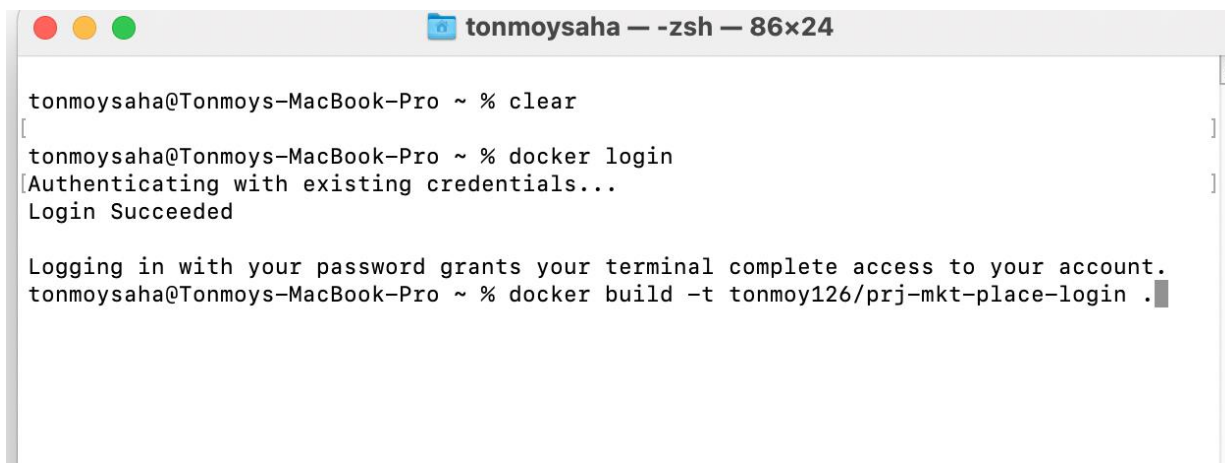
```
docker login
```

Build a local Docker image by using docker build command

```
docker build -t tonmoy126/prj-mkt-place-login.
```

Push the image in Docker hub

```
docker push tonmoy126/ prj-mkt-place-login: tag
```

A terminal window titled 'tonmoysaha -- zsh -- 86x24' showing the following commands and output:

```
tonmoysaha@Tonmoys-MacBook-Pro ~ % clear
[
tonmoysaha@Tonmoys-MacBook-Pro ~ % docker login
[Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
tonmoysaha@Tonmoys-MacBook-Pro ~ % docker build -t tonmoy126/prj-mkt-place-login .
```

Fig. 10. How to create a Docker image

```

tonmoysaha@Tonmoys-MacBook-Pro ~ % docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
tonmoy126/prj-mkt-place-login  latest      b306ed71f2a3     4 weeks ago      147MB
tonmoy126/prj-mkt-place-ord    latest      092a6dd36588     4 weeks ago      142MB
tonmoy126/prj-mkt-place-ord    <none>     5283ee7883dd     4 weeks ago      142MB
tonmoy126/prj-mkt-place-prd    latest      ae99ff58c810     4 weeks ago      142MB
tonmoy126/prj-mkt-place-prd    <none>     a067eebc44b0     4 weeks ago      142MB
tonmoy126/k8s-app-login       latest      9fb84cead35d     5 weeks ago      147MB

```

Fig. 11. All the images that have been created in the project

### 4.3.2 Using Kubernetes to Orchestrate Microservice

Create YAML file defining each POD

```

1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: webapp-login
5    labels:
6      app: webapp-login
7  spec:
8    containers:
9      - name: webapp-login
10     image: tonmoy126/prj-mkt-place-login

```

Fig. 12. YAML file for UI service of POD definition



```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: productapi
5    labels:
6      app: productapi
7  spec:
8    containers:
9      - name: webappsvc
10     | image: tonmoy126/prj-mkt-place-prd
```

Fig. 13. YAML file for Product service of POD definition

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: orderapi
5    labels:
6      app: orderapi
7  spec:
8    containers:
9      - name: webappord
10     | image: tonmoy126/prj-mkt-place-ord
```

Fig. 14. YAML file for Order service of POD definition

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4  | name: mysql
5  | labels:
6  |   app: mysql
7  spec:
8  | containers:
9  |   - name: mysql
10 |     image: mysql:8
11 |     env:
12 |     - name: MYSQL_ROOT_PASSWORD
13 |       value: password
14 |     - name: MYSQL_DATABASE
15 |       value: mkt_app_db
```

Fig. 15. YAML file for Database service of POD definition

Create a YAML file to create a service

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4  | name: k8s-login
5
6  spec:
7  | selector:
8  |   app: webapp-login
9
10 | ports:
11 |   - name: http
12 |     port: 8085
13 |     nodePort: 30080
14
15 | type: NodePort
```

Fig. 16. YAML file for UI service of Service Definition

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: productapi
5    labels:
6      app: productapi
7  spec:
8    containers:
9      - name: webappsvc
10     image: tonmoy126/prj-mkt-place-prod
```

Fig. 17. YAML file for Product service of Service Definition

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: orderapisvc
5
6  spec:
7    selector:
8      app: orderapi
9
10   ports:
11     - port: 8087
12     type: ClusterIP
```

Fig. 18. YAML file for Order service of Service Definition

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4  | name: database
5
6  spec:
7  | selector:
8  |   app: mysql
9
10 | ports:
11 | - port: 3306
12 | type: ClusterIP

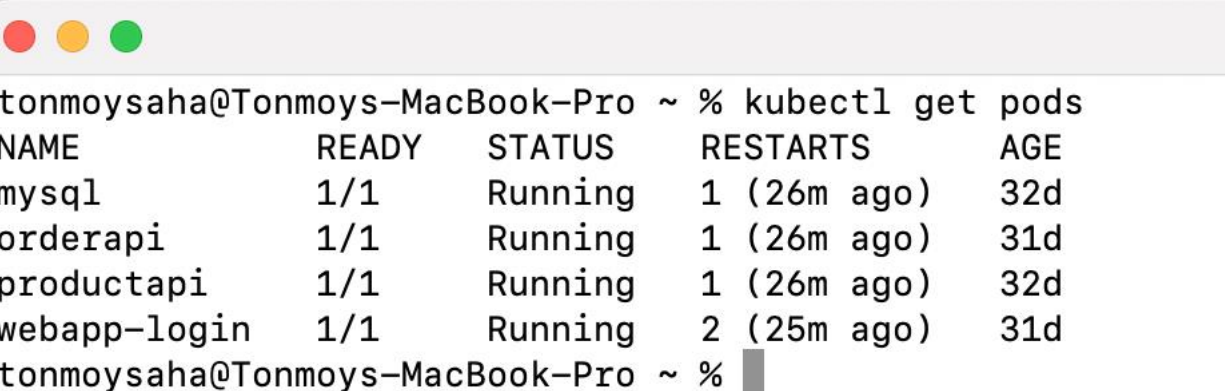
```

Fig. 19. YAML file for Database service of Service Definition

Apply the YAML file in Kubernetes to create POD and Services

After running the **kubectl apply** commands PODs and services will be created.

Figure 20 shows the PODS that are created.



```

tonmoysaha@Tonmoys-MacBook-Pro ~ % kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql         1/1     Running   1 (26m ago) 32d
orderapi      1/1     Running   1 (26m ago) 31d
productapi    1/1     Running   1 (26m ago) 32d
webapp-login  1/1     Running   2 (25m ago) 31d
tonmoysaha@Tonmoys-MacBook-Pro ~ %

```

Fig. 20. Shows all the PODs that are created in the system

```

tonmoysaha@Tonmoys-MacBook-Pro ~ % kubectl get services
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
database            ClusterIP     10.96.53.174    <none>           3306/TCP         32d
k8s-login            NodePort      10.98.236.197   <none>           8085:30080/TCP   31d
kubernetes           ClusterIP     10.96.0.1       <none>           443/TCP          47d
orderapisvc         ClusterIP     10.111.163.253 <none>           8087/TCP         31d
productapisvc       ClusterIP     10.110.112.187 <none>           8086/TCP         32d
tonmoysaha@Tonmoys-MacBook-Pro ~ %

```

Fig. 21. Shows all the Services that are created in the system

#### 4.4 Application Use Case

There are two significant categories of use cases in this application: the General User use case, and the Admin user use case. The different use cases of this application have been depicted in Figure 22.

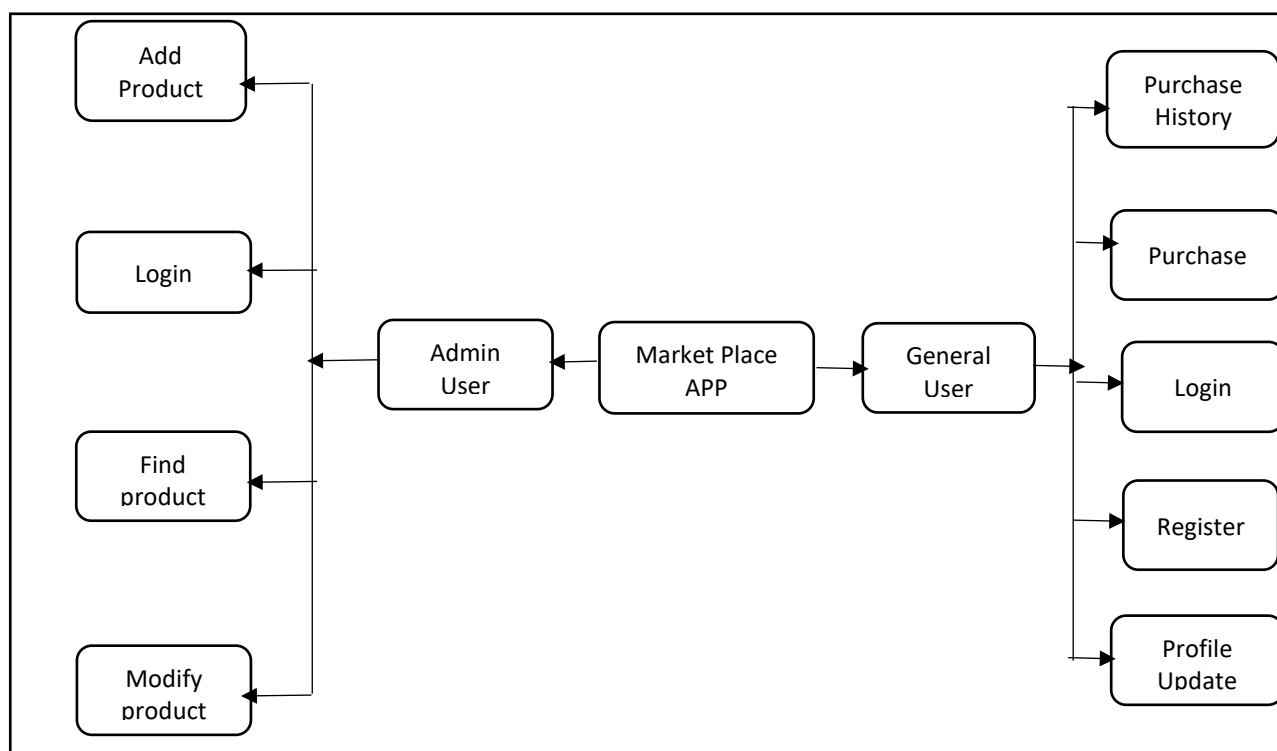


Fig. 22. Interaction diagram for Market Place Application

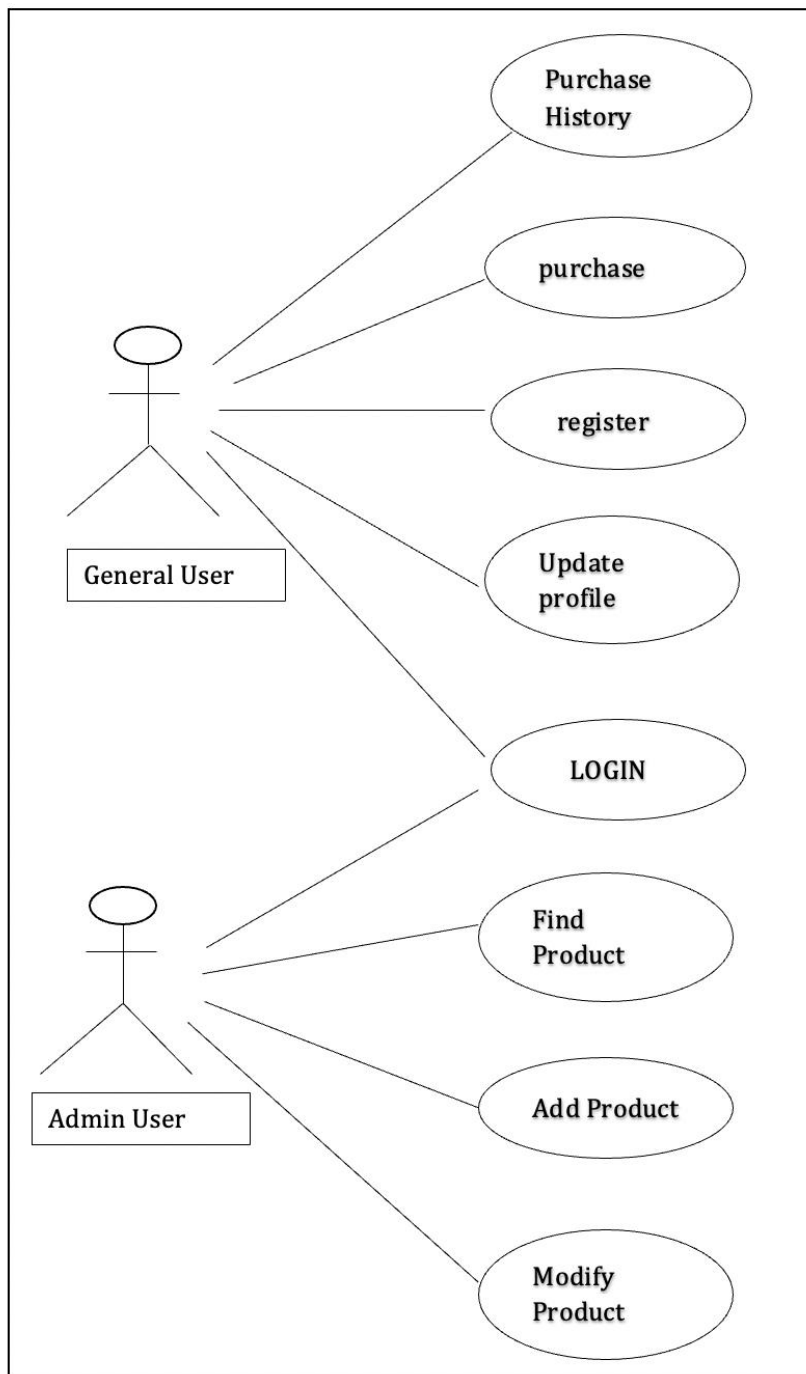


Fig. 23. Use case for the application

#### 4.4.1 Use Case - General User Login

Table 1: General User Login Use Case Description

<p>Description</p> <p>The Login Use case allows the General user to login to the Market Place App</p>
<p>Steps:</p> <p>Enter the username in the login screen</p> <p>Enter the password in the login screen</p> <p>Once the username and password are validated user is displayed a success screen. Failed login stays on the login page</p> <p>Exception:</p> <p>3.a. User entered an incorrect username or password</p> <p>3. b. User aborts and starts with the main landing page again.</p> <p>Alternate Flow:</p> <p>1. a User enters a different username.</p> <p>2. b. The user enters a different password.</p>

#### 4.4.2 Use Case - General User purchase

Table 2: General User purchase Use case Description

<p>Description</p> <p>The purchase Use case describes the purchasing of items in the Market Place APP</p>
<p>Steps</p> <p>The user selects one or more products from the displayed list of products</p> <p>The user selects the cart icon to proceed to checkout</p> <p>Users can decrease the number of selected products</p> <p>Users can see the amount purchased in the cart icon</p> <p>Users can dismiss the selected products and go back to the main page</p> <p>Users can proceed with the purchase and order of products.</p> <p>Exception:</p> <p>2. a User needs to be logged in to make the transaction</p> <p>2. b User performs the login process and proceeds to checkout</p> <p>Alternate flow:</p> <p>6. a User can dismiss the product selection</p> <p>6. b User goes back to the main landing page and re-selects products</p>



#### 4.4.3 Use Case - General User Register

Table 3: General User Register Use case Description

<p>Description</p> <p>The Register use case describes how a user can sign up for the Market place application</p>
<p>Steps</p> <p>The user selects the signup option on the main page</p> <p>The user then enters the username and password</p> <p>If the entered username is unique a new login is created for the user</p> <p>The user lands on the logged-in page</p> <p>Exception:</p> <p>3. a User enters a user id that already exists.</p> <p>3. b User is given an error message and asks the user to choose a different id.</p> <p>Alternate flow:</p> <p>1. a User reenters a unique username</p> <p>1. b. The user lands on the logged-in page.</p>

#### 4.4.4 Use Case - General User Update Profile

Table 4: General User Update Profile Use Case Description

<p>Description</p> <p>The update profile use case allows the logged-in user to modify the user profile</p>
<p>Steps:</p> <p>Logged-in users can select the “Profile” option</p> <p>The profile page loads with existing profile data</p> <p>The user can update any of the existing data</p> <p>The new data will be saved</p> <p>Exception:</p> <p>3.a. User tries to modify the username</p> <p>3. b System does not allow to modify username.</p>

#### 4.4.5 Use Case - General User Purchase History

Table 5: General User purchase History Use case Description

<p>Description</p> <p>The purchase history use case describes the past purchases made by the logged in user</p>
<p>Steps</p> <p>Logged in user selects the “purchase” option</p> <p>The list of all past purchases made by the user is displayed on the page</p> <p>Exception:</p> <p>2. a System error occurs, and no list is returned. The user returns to the main page</p>

#### 4.4.6 Use Case - Admin User Login

Table 6: Admin User Login Use Case Description

<p>Description</p> <p>The Admin User Login Use case enables the user to log in as Admin</p>
<p>Steps</p> <p>Enter the username in the login screen</p> <p>Enter the password in the login screen</p> <p>Once the username and password are validated user is displayed a success screen. Failed login stays in the login page</p> <p>Exception:</p> <p>3.a. User entered an incorrect username or password</p> <p>3. b. User aborts and starts with the main landing page again.</p> <p>Alternate Flow:</p> <p>1. a User enters a different username.</p> <p>2. b. The user enters a different password.</p>

#### 4.4.7 Use Case - Admin User Find Product

Table 7: Admin User Find Product Use Case Description

<p>Description</p> <p>The Admin user Find product use case enables the logged-in admin user to search for a product by a product id.</p>
<p>Step</p> <p>Admin user selects the “Find Product” option on the screen</p> <p>The user then enters the product id for the search</p> <p>The detail of the products is displayed on the screen</p> <p>Exception:</p> <p>3. a There is no product in the system for the searched criteria.</p> <p>3. b The search returns an empty response</p> <p>Alternative:</p> <p>2. a User enters different search criteria.</p>

#### 4.4.8 Use Case - Admin User Add Product

Table 8: Admin User Add Product Use Case Description

<p>Description</p> <p>The admin user Add Product Use case enables the admin user to add a new product to the system</p>
<p>Step</p> <p>The logged-in admin user selects the “Add Product” option on the screen</p> <p>A new form opens, directing the user to enter details of a product</p> <p>Once the user enters all the data, the user can press add to store the information</p> <p>Exception:</p> <p>3. a System fails to save the data</p> <p>Alternate flow:</p> <p>1. a User cancels and returns to the landing page.</p>

#### 4.4.9 Use Case - Admin User Modify Product

Table 9: Admin User Modify Product Use Case Description

<p>Description</p> <p>The Admin user Modify Product Use case enables the Admin user to make a change to an existing product on the system</p>
<p>Steps</p> <p>The logged-in Admin user selects on “find a product”</p>

User searches the desired product by its product Id

The user makes the required changes to the product data

The user saves the data

Exception:

4.a The system fails to save the data

Alternate Flow:

1.a User cancels the operation and returns to the landing page.

## 4.5 Application Screenshots

The different features and functionalities of the Market Place application are present below in screenshots:

### 4.5.1 Landing/Main Page

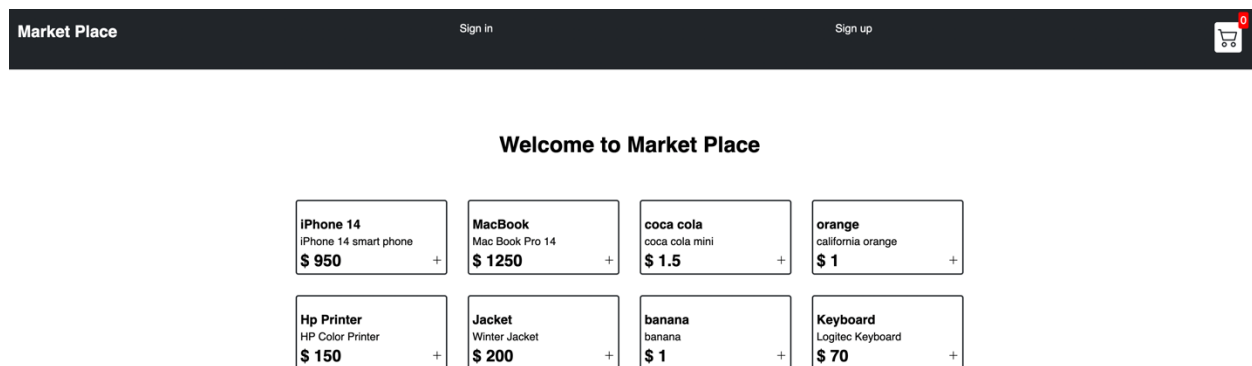


Fig. 24. Landing Screen of The Market Place App

## 4.5.2 The Login Page

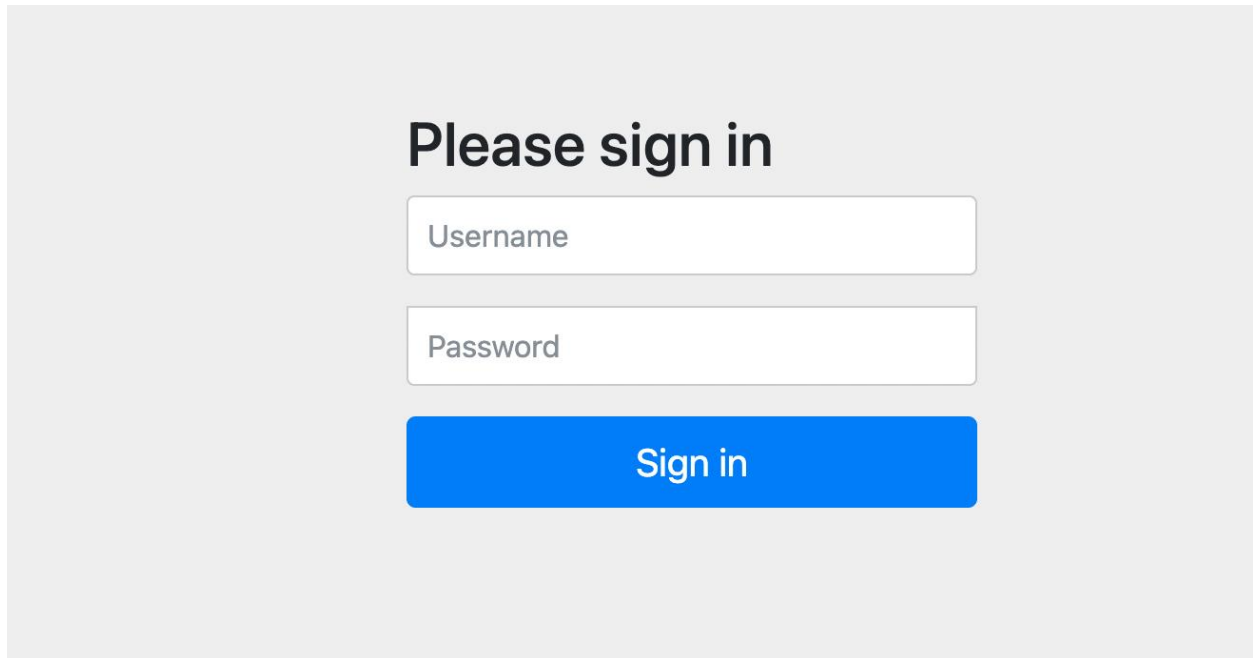


Fig. 25. Login Screen of The Market Place App

## 4.5.3 Admin Home

Product Id	Product name	Product Desc	Qty	Price	Rating
1	iPhone 14	iPhone 14 smart phone	10	950	5
2	MacBook	Mac Book Pro 14	30	1250	5
3	coca cola	coca cola mini	50	1.5	4
4	orange	california orange	60	1	3
5	Hp Printer	HP Color Printer	20	150	4
6	Jacket	Winter Jacket	5	200	5
7	banana	banana	60	1	4
8	Keyboard	Logitech Keyboard	3	70	5

Fig. 26. Admin Home Screen of The Market Place App

#### 4.5.4 Add Product

Market Place Logout testAdmin Admin User

[Back to Home](#) [Find Product](#) [Add Products](#)

### Maintain Product

product Id:

Product name:

Product Desc:

Product Quantity:

Product price:

Product rating:

Product image:

[Add](#)

Fig. 27. Add Product Screen of The Market Place App

#### 4.5.5 Find product

Market Place Logout testAdmin Admin User

[Back to Home](#) [Find Product](#) [Add Products](#)

### Maintain Product

Product ID:

Product Name:

[Find](#)

Fig. 28. Find the Product Screen of The Market Place App

#### 4.5.6 Modify Product

Market Place Logout testAdmin Admin User

[Back to Home](#) [Add Products](#)

### Maintain Product

product Id:

Product name:

Product Desc:

Product Quantity:

Product price:

Product rating:

Product image:

[Add](#)

Fig. 29. Modify the Product Screen of The Market Place App



### 4.5.7 User Home

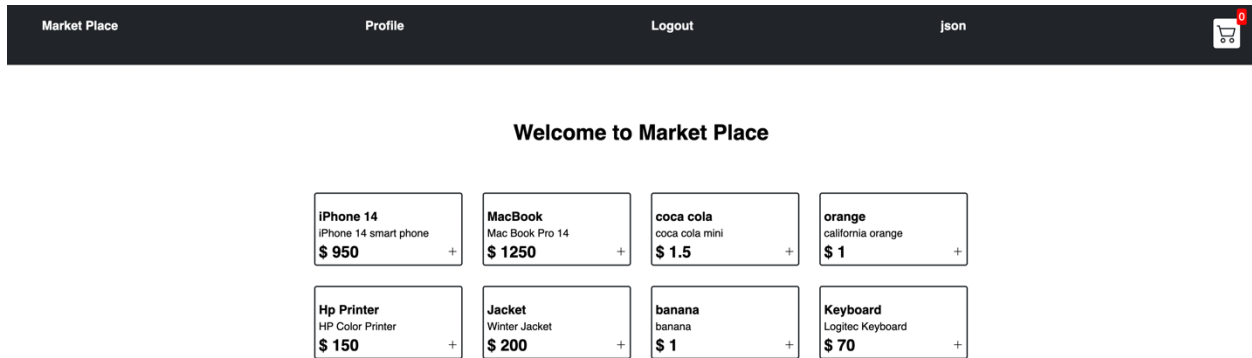


Fig. 30. User Home Screen of The Market Place App

### 4.5.8 User Profile Update

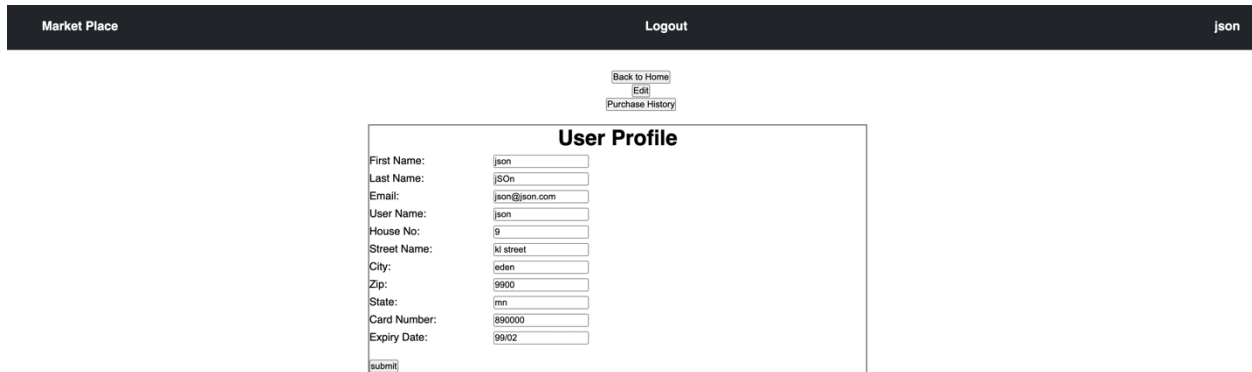


Fig. 31. User Profile Update Screen of The Market Place App

### 4.5.9 Purchase History

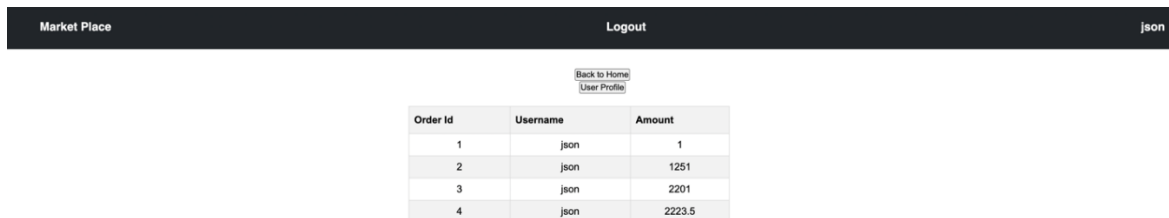


Fig. 32. Purchase History Screen of The Market Place App

## **Chapter 5: Demonstrating the advantages of Microservice**

The previous Chapters have described the Market Place Application using microservice design architecture. This chapter will highlight the benefits of this architecture.

### **5.1 Ease of Deployment and Maintenance**

The individual services here, The UI service, Product service, and Order service, have separate java codes. These services are built individually, and three docker images are prepared. If any of the business requirements on any of the feature changes, that service is coded for the new change, a new jar file is generated, and a new docker image is prepared. Then the new image is deployed via Kubernetes. While performing the above-mentioned steps, no other services are touched. The change is confined to one service, and a deployment to the production environment is done only for one service. This simplifies the process of adopting new change.

### **5.2 High Elasticity and Selectively Scalable**

Elasticity and Scalability ensure that an application can handle an increased amount of load at any time. One example of this is, during the holiday season, there will be an increased number of purchases in e-commerce applications. The purchase feature in the application will see increased traffic. An elastic application should be able to scale its resource and provide service to this increased load. If the application is monolithic, then the whole application needs to scale up. You must add resources like memory disk space and CPU to handle an increased load. But in a microservice application, selective scaling is possible. Here only the Order service needs to scale up. This scaling up could be done by increasing the number of Pods in the Order service. The rest of the services can remain as it

is. So, the number of resources (memory, CPU) needed to handle the same amount of increased load is comparatively very less.

### **5.3 Allowing the Co-existence of Different Versions**

In a production-grade application, whenever a new feature is delivered, it is rolled out to the general public in a phased manner. Doing this in a monolithic application is very difficult. In a monolithic application, usually, a different cluster is maintained, and into that cluster, the whole application with the new feature is deployed. As the application is always bundled in one single war file in a monolithic application, the entire application needs to be deployed in that cluster. It is similar to having two production environments.

The same could be achieved in microservice design in a relatively more straightforward and cost-effective manner. The service that has this new feature is versioned as v1 and v2. The latest and old versions of the service, for example, the product service, are deployed in production. The gateway will route traffic either to v1 or v2 depending on some condition. In this design, only one service will have a different version. The rest of the application will be only one. Thus, it consumes less infrastructure and hence is more cost-effective.

### **5.4 Better Quality of Service**

Microservice applications are more resilient and provide better service due to infrastructure failure. Any system will have its share of downtime. Now the important question here is, what is the impact on the user experience? In a typical monolithic application, downtime means the application is unavailable to the user, and the user cannot do anything on the application.

In a microservice application like the one we have discussed here, the Market Place e-commerce application. The user impact due to downtime is very localized. For example, if

the Order service is down. The user will not be able to purchase any new items on the application. However, the user would still be able to login into the application and update user profile-related information. Browse through the product catalogs. In this Market Place application, each feature is provided via a different microservice. So, when the Order service is down, the Product and UI service is still up. The application is never down. Thus, providing a better quality of service.

### **5.5 Co-development and speed of delivery**

In the traditional application, the code base for the entire application is one. Multiple developers working on a large application become complex. It comes with extra overhead of code merge, conflict resolution, and regression testing. This adds up complexities to software development. Making any change to the existing application requires more time and effort. Sometimes the different features are so tightly coupled that making a change without impacting the other feature becomes impossible.

## Chapter 6: Conclusion

The microservice architecture has evolved from the current business needs and best practices to solve industry-level problems that an application has to face continuously. With the proper support from technological advancement in application development, today's microservice design pattern is an efficient solution. The containerization technique from Docker and the orchestration mechanism from Kubernetes made this design very popular in the software industry.

The main objective of this project was to describe the microservice architecture in detail and introduce the relevant technology in this field. A hands-on project was created to replicate all the steps in developing a microservice application. While doing so, I learned the different benefits of microservices architecture design. I have highlighted some significant advantages of microservice over monolithic applications.

There is still scope for further addition to this project. Other features in Kubernetes could be implemented. DevOps could be involved in automatic deployment. A logging service could be added for audit and monitoring.

## References

- [1] “Spring Boot.” Spring.io. <https://spring.io/projects/spring-boot> (accessed Oct. 10, 2022)
- [2] R. V. Rajesh, *Spring 5.0 Microservices*, 2<sup>nd</sup> ed. Birmingham, UK: Packt, 2017
- [3] J. Carnell, *Spring Microservices in Action*, Manning Publications, 2017.  
<https://learning.oreilly.com/library/view/spring-microservices-in/9781617293986/>  
(accessed Oct. 12, 2022)
- [4] J. Krochmalski, *Docker and Kubernetes for Java Developers*. Packt Publishing, 2017.  
<https://learning.oreilly.com/library/view/docker-and-kubernetes/9781786468390/>  
(accessed Oct. 12, 2022)
- [5] P. Fisher, “Docker in Motion.” O’reilly. <https://learning.oreilly.com/videos/docker-in-motion/10000MNLV201711/>(accessed Oct 15, 2022)
- [6] “Learn Kubernetes.” Kubernetes.io. <https://kubernetes.io/docs/home/> (accessed Oct 5, 2022)
- [7] “Docker.” Docker.com. <https://www.docker.com/get-started>  
(accessed June 10, 2022)
- [8] “JavaScript.” MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (accessed June 10, 2022)
- [9] “HTML, CSS and JavaScript.” W3Schools. <https://www.w3schools.com/> (accessed June 10, 2022)