

Reactive Spring Boot Application For Product Variant Using Kafka And Cassandra Database

Project report submitted in partial fulfilment of the requirement for
the degree of Bachelor of Technology

In

Computer Science and Engineering

By:

Aditya Kumar Singh (171289)

Under the supervision

of

Mr. Aman Sinha

(Tech Lead, ZopSmart Technology)

To



Department of Computer Science & Engineering and Information Technology

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT, SOLAN – 173234, HIMACHAL PRADESH**

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled “Reactive Spring Boot Application For Product Variant Using Kafka And Cassandra Database” in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science submitted in the Department of Computer Science Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from February 2021 To May 2021 under the supervision of Mr. Aman Sinha.

The matter embodied in the report has not been appeased for the award of any other degree or diploma.



Aditya Kumar Singh (171289)

This is to verify that the above statement made by candidates is true to the best of my knowledge.



Mr. Aman Sinha

Tech Lead

ZopSmart Technology

Dated:- 22-05-2021

ACKNOWLEDGEMENT

We have taken efforts to do this project. We wish to express our sincere gratitude to Mr. Aman Sinha, Tech Lead, ZopSmart Technology for constantly monitoring and guiding me to the right path in terms of the project. She constantly helped us in our research and the project wouldn't be possible without his constant support.

Secondly, I would also like to thank Lab assistant who helped me a lot in finalizing this project within the limited time frame.

LIST OF CONTENTS

CHAPTER-1 INTRODUCTION	1
1.1 Introduction	2
1.2 Problem statement	3
1.3 Objectives	5
1.4 Methodology	5
1.5 Organization:	7
CHAPTER-2 LITERATURE SURVEY	8
CHAPTER-3 SYSTEM DEVELOPMENT.....	12
3.1 Reactive Programming	14
• Basics.....	11
• Reactive Systems.....	15
• Reactive Stream Specification.....	15
3.2 Spring Boot	17
• Experimental setup	18
3.3 Rest Api	20
3.4 Apache Kafka.....	22
3.5 Apache Cassandra.....	32
3.6 Model Development.....	39
3.6 Features.....	41
CHAPTER-4 PERFORMANCE ANALYSIS.....	42
CONCLUSION	48
FUTURE WORK.....	48
REFERENCE.....	49

LIST OF FIGURES

Figure 1	Cloud Computing Services.....	12
Figure 2	Docker architecture	13
Figure 3	Reactive Event Stream.....	14
Figure 4	Spring Initializer	19
Figure 5	URI.....	21
Figure 6	Point to Point Messaging System	24
Figure 7	Publish Subscribe Messaging System.....	24
Figure 8	Kafka	26
Figure 9	Cluster Diagram of Kafka	29
Figure 10	Node Replication	34
Figure 11	Cassandra Write Operation.....	36
Figure 12	Read Operation	37
Figure 13	Flow Chart	39
Figure 14	Schema diagram	40
Figure 15	Spring Boot Application	42
Figure 16	Kafka ZooKeeper	43
Figure 17	Kafka Server.....	44
Figure 18	Kafka Consumer	44
Figure 19	Apache Server Image.....	45
Figure 20	Docker Image.....	46
Figure 21	Building container	46
Figure 22	Database Image	47
Figure 23	Database runtime	47

Abstract

The Zopsmart Smart Store is a platform to help businesses setup their own customized e-commerce website. It has two variants: Smart Store Eazy, and Smart Store Enterprise Edition. As the name suggests for both Smart Store Enterprise Edition contains many features that make it modular and scalable, it has minimal integration, you can operate it for multiple stores, any many more customer friendly features. Whereas Smart Store Eazy is a lighter version of the Enterprise Edition and is intended for small grocery stores. So, when a client uses it to build his/her own e-commerce platform, it must be ensured that they get access the features they need, for that multiple extensions have been provided so that the client can curate their website as per their own requirement. When it comes to clients from abroad, or any business that has to extend abroad, there arises the need of multi-lingual support on the website. This helps the customers search in their respective languages and receive the search results in their respective language. But nothing is so easy when it comes to integrate this feature. A separate technology needs to be added in order to handle this and in order to avoid uncontrolled calls to the database. Usage of other technologies (such as Google APIs) helps in translation of the content as per the language variation.

CHAPTER 1

INTRODUCTION

Product variants are used to manage products having different variations, like size, colour, etc. It allows managing the product at the template level (for all variations) and at the variant level (specific attributes). As an example, a company selling t-shirts may have the following product: Levi's T-shirt .It will have Sizes: S, M, L, XL, XXL , Colours: Blue, Red, White, Black etc.

In this example, Levi's T-Shirt is called the product template and T-Shirt, S, Blue is a variant. Sizes and color are attributes.

The above example has a total of 20 different products (5 sizes x 4 colors). Each one of these products has its own inventory, sales, etc. An E-commerce website requires a multilingual support because it eases the usage and searching of products for customers using the platform. Also, it as an important aspect if your business is purely internet based. An intuitive user interface is a hallmark of any decent business. But in order for the frontend of applications to work smoothly, you must also consider the backend. Backend development, also called server-side development, handles the behind-the-scenes functions of web development – things like interactions with databases, authorizing users and routing URLs.

Reactive Programming (RP) is a programming model that is designed to cope with asynchronous events (data streams) and the specific act of producing a change, in other words, it means that modifications are implemented to the execution environment in an effective ceratin order. Take a look at the sequences of events in real life in order to have a full understanding of the reactive programming Java paradigm.Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

Cassandra is a distributed database management system designed for handling a high volume of structured data across commodity servers. Cassandra handles the huge amount of data with its distributed architecture. Data is placed on different machines with more than one replication factor that provides high availability and no single point of failure.

There are lots of great reasons why you should use reactive programming as a business or developer.

Here are the major ones to think about.

1. Improves user experience - this is at the very heart of why you should be using reactive programming for your apps or websites. The asynchronous nature of FRP means that whatever you program with it will offer a smoother, more responsive product for your users to interact with.
2. Easy to manage - one big bonus with reactive programming is that it is easy to manage as a developer. Blocks of code can be added or removed from individual data streams which means you can easily make any amendments needed via the stream concerned.
3. Simpler than regular threading - FRP is less hassle than regular threading due to the way it allows you to work on the data streams. Not only is this true for basic threading

in an application but also for more complex threading operations you may need to undertake.

Problem statement

- **Thread Per Request Model:-**

The application will only be able to handle a number of concurrent requests that equals the size of the thread pool. It is possible to configure the size of the thread pool, but since each thread reserves some memory (typically 1MB), the higher thread pool size we configure, the higher the memory consumption. If the application is designed according to a microservice based architecture, we have better possibilities to scale based on load, but a high memory utilization still comes with a cost. that the greatest advantage of cloud computing. It encourages you to spare important capital expense because it needn't trouble with any actual Instrumentality ventures.

- **Waiting for I/O operation:-**

Same type of waste also occurs while waiting for other types of I/O operations to complete such as a database call or reading from a file. In all these situations the thread making the I/O request will be blocked and waiting idle until the I/O operation has completed, this is called blocking I/O. Such situations where the executing thread gets blocked, just waiting for a response, means a waste of threads and therefore a waste of memory.

- **Response Time:-**

Another issue with traditional imperative programming is the resulting response times when a service needs to do more than one I/O request. For example, service A might need to call service B and C as well as do a database lookup and then return some aggregated data as a result. This would mean that service A's response time would besides its own processing time be a sum of:

- response time of service B (network latency + processing)
- response time of service C (network latency + processing)
- response time of database request (network latency + processing)

- **Overwhelming the client:-**

Another type of problem that might occur in a microservice landscape is when service

A is requesting some information from service B, let's say for example all the orders placed during last month. If the amount of orders turns out to be huge, it might become a problem for service A to retrieve all this information at once. Service A might be overwhelmed with the high amount of data and it might result in for example an out of memory-error.

- **On Demand Self Service**

Developer not have to be compelled to worry regarding the resources. Resources square measure created on the market to the user on AN “as needed” basis. instead of all quickly , on-demand computing permits cloud hosting firms to supply their purchasers with access to computing resources as they become necessary

The different issues described above are the issues that reactive programming is intended to solve. In short, the advantages that comes with reactive programming is that we:

- move away from the thread per request model and can handle more requests with a low number of threads.
- prevent threads from blocking while waiting for I/O operations to complete.
- make it easy to do parallel calls.
- support “back pressure”, giving the client a possibility to inform the server on how much load it can handle.

Needs of Microservices

1. Continuous Delivery

Microservices provide the ideal architecture for continuous delivery. With microservices, each application resides in a separate container along with the environment it needs to run. Because of this, each application can be edited in its container without the risk of interfering with any other application.

This means zero downtime for users, simplified troubleshooting, and no disruption even if a problem is identified. The safe and rapid changes allowed by microservice architecture make it possible to update software fast enough to put the “continuous” in continuous delivery. By keeping disruption to a minimum, microservice architecture lets you update rapidly without inconveniencing customers.

2. Maximize Deployment Activity

Microservice architecture allows you to maximize deployment velocity and

application reliability by helping you move at the speed of the market. Since applications each run in their own containerized environment, applications can be moved anywhere without altering the environment. If an application works in development, it will work for the customer. This speeds up time to market and increases product reliability

3. Faster innovation to adapt to changing market conditions

Microservices can also help you adapt more quickly to the changing market conditions. Because microservices allow applications to be updated and tested quickly, you can follow market trends and adapt your products faster.

Microservices also give you an edge when it comes to innovation, since developers can experiment on applications without fear of causing problems elsewhere. In today's rapidly changing market, getting an edge on innovation helps you maintain your current revenue streams while driving new revenue

Objectives

To make a Spring Boot non-blocking applications that are asynchronous and event-driven and require a small number of threads to scale. A key aspect of that definition is the concept of backpressure which is a mechanism to ensure producers don't overwhelm consumers .

Methodology

It is a system of broad principles or rules from which specific methods or procedures may be derived to interpret or solve different problems within the scope of a particular discipline. Unlike an algorithm, a methodology is not any formula but it is a set of practices.

- **Amazon Web Services**

Services that can be used are:-

- Amazon Elastic Compute Cloud (Amazon EC2) to run Linux or Windows based servers
- Elastic Load Balancing (ELB) to load balance and distribute the web traffic
- Amazon Elastic Block Store (Amazon EBS) or Amazon Elastic File System (Amazon EFS) to store static content.
- Amazon Virtual Private Cloud (Amazon VPC) to deploy Amazon EC2 instances. Amazon VPC is your isolated and private virtual network in the AWS Cloud and gives you full control over the network topology, firewall configuration, and routing rules.
- Web servers can be spread across multiple Availability Zones for high availability, even if an entire data center were to be down.
- AWS Auto Scaling automatically adds servers during high traffic periods and scales back when traffic decreases

- **Docker**

Docker may be a instrumentation platform that packages your application and every one its dependencies along within the sort of a dockhand container to confirm that your application works seamlessly in any atmosphere. dockhand maybe a powerful tool for making and deploying applications. It simplifies rolling out applications across multiple systems and may be a useful gizmo for desegregation new technologies. Associate in Nursing application that runs victimisation dock- hand can start off identical each time occasion anytime on every system. this suggests that if the applying works on your native laptop, it'll work anyplace that supports dockhand.

- **Docker Image**

A dockhand image may be a read-only example that contains a collection of Direc- tions for making a instrumentality that may run on the dockhand platform. It prov- ides a convenient thanks to package up applications and preconfigured server environments, that you'll be able to use for your own non- public use or share pub- lically with different dockhand users.

- **Docker Containers**

Containers are the style of software virtualization. one instrumentality may well be accustomed run something from a tiny low micro-service or code method to a bigger application. within a instrumentality are all the mandatory executables, computer code, libraries, and configuration files. Compared to server or machine virtualization approaches, however, containers don't contain software pictures. This makes them a lot of light-weight and moveable, with considerably less overhead. In larger application deployments, multiple instrumentality is also deployed collectively or a lot of container clusters. Such clusters may well be managed by a instrumentality adapter like Kubernetes.

1.5 Organization

This report has been organized with the following chapters:

Chapter 2: In this chapter the previous end related work done in the development of this project were described with their methodology and architecture proposed by the authors.

Chapter 3: In this chapter we see how we work on system design. Also described Spring Boot , Apache Kafka , Reactive Programming , REST and Cassandra Database which will be used in the development of spring boot application.

Chapter 4: Discusses about the result and Screenshots.

Chapter 5: Concludes the project and gives suggestions for future work.

CHAPTER 2

LITERATURE SURVEY

[1] Dr. Mitali Gupta [1] conducted the study of Reactive app that are available in India and concluded that only the application that were successful have good user interface and shows the image in a way that sticks in the mind of user so he or she will buy that. In short a good user interface application and low latency application gives best user experience .

[2] Shantashree Das, Debomalya Ghose [2] conducted study on influence of online food delivery application on the operation on business the paper discuss about how human behaviour are changing and they are more inclined towards ordering food online instead of going to restaurant and also provided some solution in order to operate business in more efficient manner. So Reactive Spring boot is making better user experience

[3] P.Nagendra Babu , M.Chaitanya Kumari , S.Venkat Mohan [3] has worked on computation of cloud computing , data access and storage services that do not require end user knowledge of the physical location and configuration of the system that delivers the services.

SAAS	PAAS	IAAS
Software As A Service	Platform As A Service	Infrastructure As A Service
<ul style="list-style-type: none"> ✓ Government Applications ✓ Communications ✓ Productivity tools 	<ul style="list-style-type: none"> ✓ Application Development ✓ Security Services ✓ Database Management 	<ul style="list-style-type: none"> ✓ Server ✓ Network ✓ Storage
Examples: <ul style="list-style-type: none"> ✓ Oracle ✓ SalesForce.com ✓ LinkedIn ✓ Google Apps 	Examples: <ul style="list-style-type: none"> ✓ Microsoft Azure ✓ GAE 	Examples: <ul style="list-style-type: none"> Amazon EC2 Verizon Terremark

Figure 1 Cloud Computing Services

Concluded that however Cloud is employed for organizations and the way the cloud is employed to Store , retrieve and modify the info while not physical instrumentation .Cloud computing is that the quickest new paradigm for delivering on demand services over web and might be represented as central Software system.Cloud computing describes a brand new supplement ,

consumption and delivery model for IT services supported web protocols and it generally involves provisioning of dynamically ascendable and infrequently virtualized resources .

[4] BabakBashari Rad, Harrison John Bhatti and Mohammad Ahmadi [4] worked on Docker which provide some facilities, which are useful for developers and administrators. It is an open platform can be used for building ,distributing, and running applications in a portable , lightweight runtime and packaging tool, known as Docker Engine. It also provide Docker Hub, which is a cloud service for sharing applications. Costs can be reduced by replacing traditional virtual machine with docker container. It excellently reduces the cost of re-building the cloud development platform.

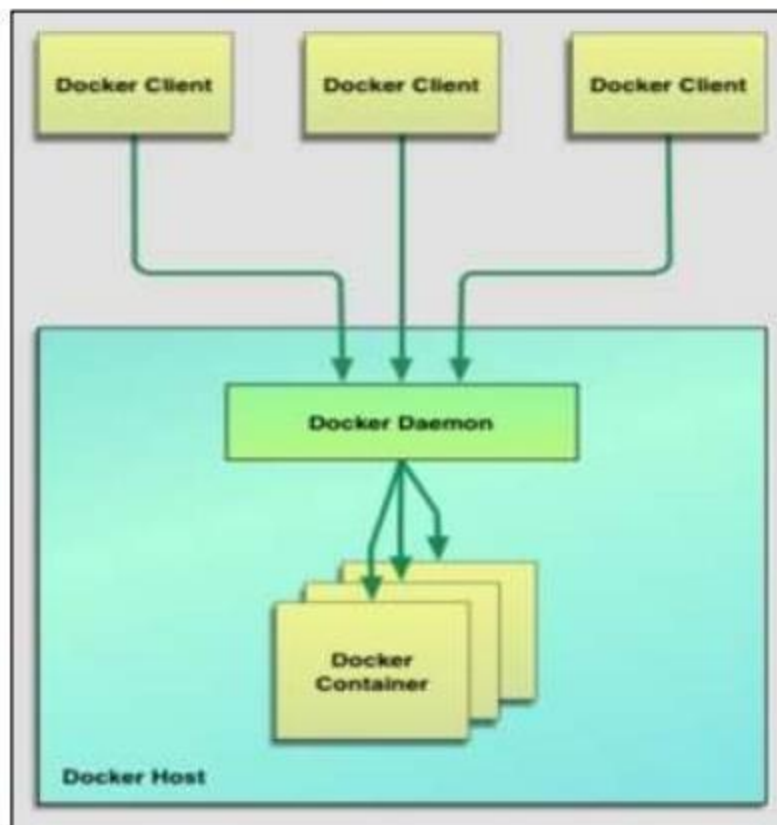


Figure 2 Docker architecture

The conclusion is Docker automates the applications when they are containerized. An extra layer of docker engine is added to the host operating system. The performance of docker is faster than virtual machines as it has no guest operating system and less resource overhead.

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 Reactive Programming

- Basics

In short: by programming with asynchronous data streams. Let's say service A wants to retrieve some data from service B. With the reactive programming style approach, service A will make a request to service B which returns immediately (being non-blocking and asynchronous). Then the data requested will be made available to service A as a data stream, where service B will publish an `onNext`-event for each data item one by one. When all the data has been published, this is signalled with an `onComplete` event. In case of an error, an `onError` event would be published and no more items would be emitted.

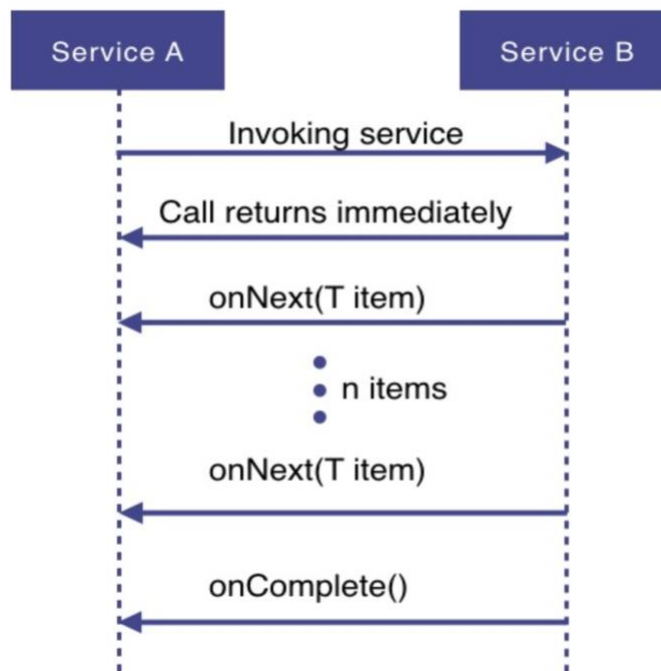


figure 3 : Reactive Event Stream

Reactive programming uses a functional style approach (similar to the Streams API), which gives the possibility to perform different kinds of transformations on the streams. A stream can be used as an input to another one. Streams can be merged, mapped and filtered .

- **Reactive Systems**

Reactive programming is an important implementation technique when developing “reactive systems”, which is a concept described in the “Reactive Manifesto”, highlighting the need for modern applications to be designed to be:

1. Responsive (responding in a timely manner)
2. Resilient (staying responsive also in failure situations)
3. Elastic (staying responsive under varying workload)
4. Message Driven (relying on asynchronous message passing)

Building a reactive system means to deal with questions such as separation of concerns, data consistency, failure management, choice of messaging implementation etc. Reactive programming can be used as an implementation technique to ensure that the individual services use an asynchronous, non-blocking model, but to design the system as a whole to be a reactive system requires a design that takes care of all these other aspects as well.

- **Reactive Streams Specification**

As time went on, a standardisation for Java was developed through the Reactive Streams effort. Reactive Streams is a small specification intended to be implemented by the reactive libraries built for the JVM. It specifies the types to implement to achieve interoperability between different implementations. The specification defines the interaction between asynchronous components with back pressure. Reactive Streams was adopted in Java 9, by the Flow API . The purpose of the Flow API is to act as an interoperation specification and not an end-user API like RxJava.

The specification covers the following interfaces:

Publisher : This represents the data producer/data source and has one method which lets the subscriber register to the publisher.

```
public interface Publisher<T> {  
    public void subscribe(Subscriber<? super T> s);  
}
```

Subscriber : This represents the consumer and has the following methods:

```
public interface Subscriber<T> {  
    public void onSubscribe(Subscription s);  
    public void onNext(T t);  
    public void onError(Throwable t);  
    public void onComplete();  
}
```

- onSubscribe is to be called by the Publisher before the processing starts and is used to pass a Subscription object from the Publisher to the Subscriber^[LSEP]
- onNext is used to signal that a new item has been emitted^[LSEP]
- onError is used to signal that the Publisher has encountered a failure and no more items will be emitted
- onComplete is used to signal that all items were emitted successfully^[LSEP]

Subscription : The subscriptions holds methods that enables the client to control the Publisher's emission of items (i.e. providing backpressure support).

```
public interface Subscription {  
    public void request(long n);  
    public void cancel();  
}
```

- request allows the Subscriber to inform the Publisher on how many additional elements to be published

- cancel allows a subscriber to cancel further emission of items by the Publisher.

Processor : If an entity shall transform incoming items and then pass it further to another Subscriber, an implementation of the Processor interface is needed. This acts both as a Subscriber and as a Publisher.

```
public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {  
    }  
}
```

3.2 Spring Boot

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

How does it work ?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using `@EnableAutoConfiguration` annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains `@SpringBootApplication` annotation and the main method.

Spring Boot automatically scans all the components included in the project by using `@ComponentScan` annotation.

Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include `spring-boot-starter-data-jpa` dependency in your project.

Note that all Spring Boot starters follow the same naming pattern `spring-boot-starter-*`, where `*` indicates that it is a type of the application.

Auto Configuration

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add `@EnableAutoConfiguration` annotation or `@SpringBootApplication` annotation to your main class file. Then, your Spring Boot application will be automatically configured.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
```

@EnableAutoConfiguration

```
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Spring Boot Application

The entry point of the Spring Boot Application is the class contains `@SpringBootApplication` annotation. This class should have the main method to run the Spring Boot application. `@SpringBootApplication` annotation includes Auto- Configuration, Component Scan, and Spring Boot Configuration.

If you added `@SpringBootApplication` annotation to the class, you do not need to add the `@EnableAutoConfiguration`, `@ComponentScan` and `@SpringBootConfiguration` annotation. The `@SpringBootApplication` annotation includes all other annotations.

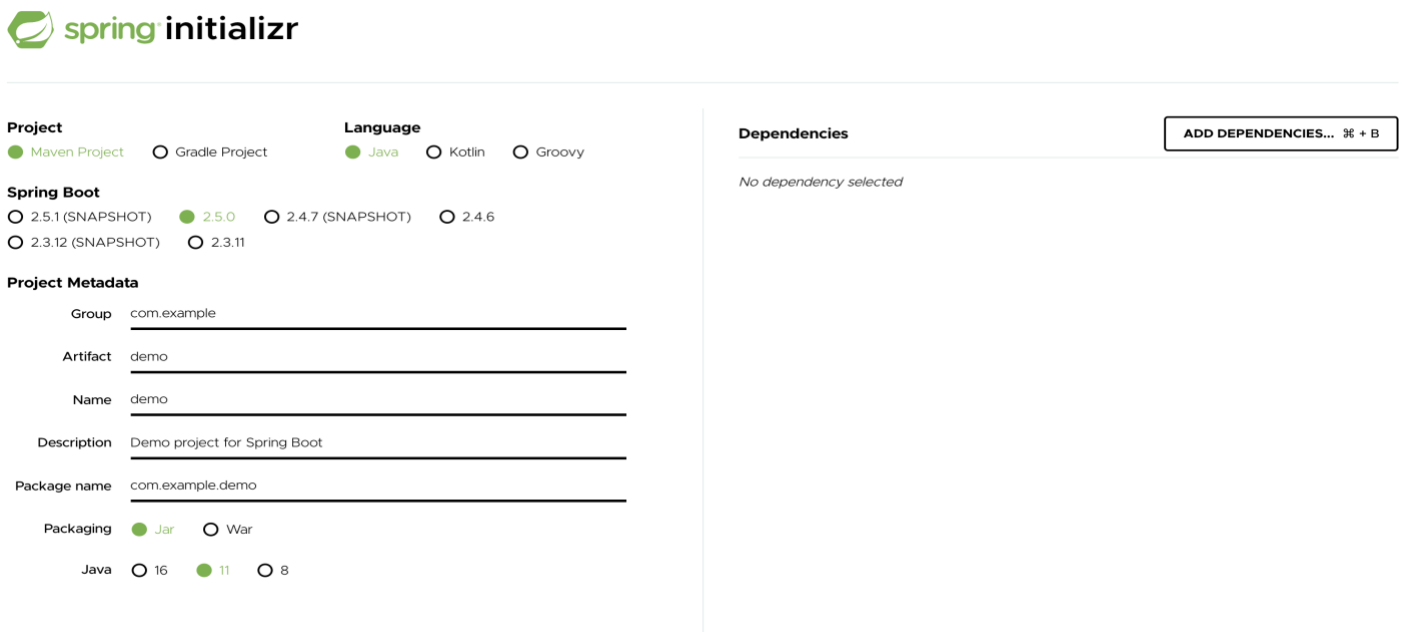
Component Scan

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the `@ComponentScan` annotation for your class file to scan your components added in your project.

Spring Initializer

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page www.start.spring.io and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.

Observe the following screenshot that shows an example where we added the **spring-boot-starter-web** dependency to write REST Endpoints.



The screenshot displays the Spring Initializer web interface. At the top left is the 'spring initializr' logo. The form is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for versions '2.5.1 (SNAPSHOT)', '2.5.0' (selected), '2.4.7 (SNAPSHOT)', '2.4.6', '2.3.12 (SNAPSHOT)', and '2.3.11'; 'Project Metadata' with text input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo); and 'Packaging' with radio buttons for 'Jar' (selected) and 'War'. Below 'Packaging' are radio buttons for 'Java' versions '16', '11' (selected), and '8'. On the right side, there is a 'Dependencies' section with the text 'No dependency selected' and a button labeled 'ADD DEPENDENCIES... ⌵ + B'.

figure 4 : Spring initializr

Properties File

Properties files are used to keep ‘N’ number of properties in a single file to run the application in a different environment. In Spring Boot, properties are kept in the application.properties file under the classpath.

The application.properties file is located in the src/main/resources directory.

YAML File

Spring Boot supports YAML based properties configurations to run the application. Instead of application.properties, we can use application.yml file. This YAML file also should be kept inside the classpath.

3.3 REST API

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

- **GET** – Provides a read only access to a resource.
- **POST** – Used to create a new resource.
- **DELETE** – Used to remove a resource.
- **PUT** – Used to update a existing resource or create a new resource.

Restful Webservices

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Sr.No.	URI	HTTP Method	POST body	Result
1	/UserService/users	GET	empty	Show list of all the users.
2	/UserService/addUser	POST	JSON String	Add details of new user.
3	/UserService/getUser/:id	GET	empty	Show details of a user.

figure 5 : URI

What is a Resource?

REST architecture treats every content as a resource. These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data. REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource where Text, JSON, XML. The most popular representations of resources are XML and JSON.

Representation of Resources

A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database. Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

For example, in Restful Web Services – First Application chapter, a user is a resource which is represented using the following XML format –

```
<user>
  <id>1</id>
  <name>Mahesh</name>
  <profession>Teacher</profession>
</user>
```

The same resource can be represented in JSON format as follows

```
{
  "id":1,
  "name":"Mahesh",
  "profession":"Teacher"
}
```

Good Resources Representation

REST does not impose any restriction on the format of a resource representation. A client can ask for JSON representation whereas another client may ask for XML representation of the same resource to the server and so on. It is the responsibility of the REST server to pass the client the resource in the format that the client understands.

Following are some important points to be considered while designing a representation format of a resource in RESTful Web Services.

- Understandability – Both the Server and the Client should be able to understand and utilize the representation format of the resource.^{[1][SEP]}

- Completeness – Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.

3.4 Apache Kafka

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

What is a Messaging System?

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system. Two types of messaging patterns are available – one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow pub-sub.

Point to Point Messaging System

In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.

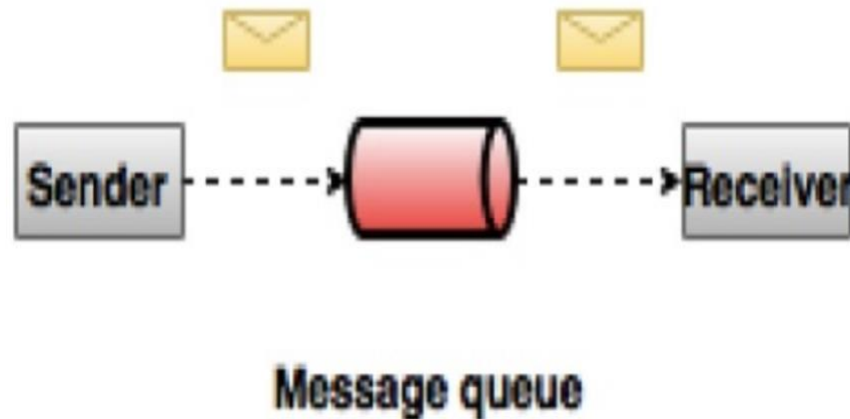


figure 6 : Point to Point Messaging System

Publish-Subscribe Messaging System

In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

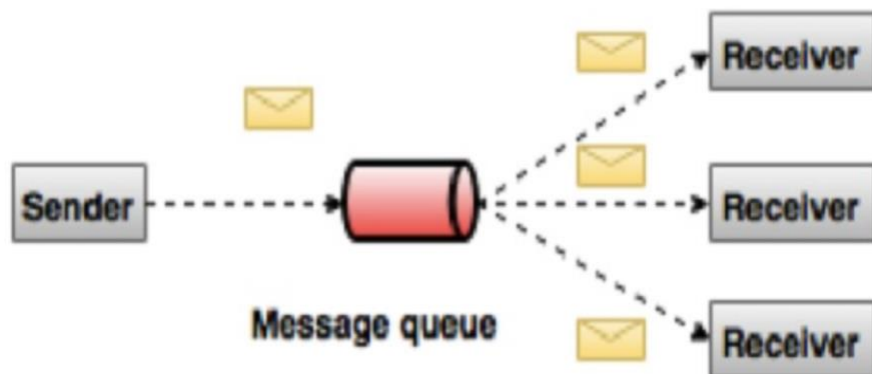


figure 7 : Publish-Subscribe Messaging System

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

Benefits

Following are a few benefits of Kafka –

- Reliability – Kafka is distributed, partitioned, replicated and fault tolerance.
- Scalability – Kafka messaging system scales easily without down time.
- Durability – Kafka uses “Distributed commit log” which means messages persists on disk as fast as possible, hence it is durable.
- Performance – Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

Use Cases

Kafka can be used in many Use Cases. Some of them are listed below –

- Metrics – Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
- Log Aggregation Solution – Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.
- Stream Processing – Popular frameworks such as Storm and Spark Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications. Kafka’s strong durability is also very useful in the context of stream processing.

Need for Kafka

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It has the ability to handle a large number of diverse consumers. Kafka is very fast, performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

Before moving deep into the Kafka, you must aware of the main terminologies such as topics, brokers, producers and consumers. The following diagram illustrates the main terminologies and the table describes the diagram components in detail.

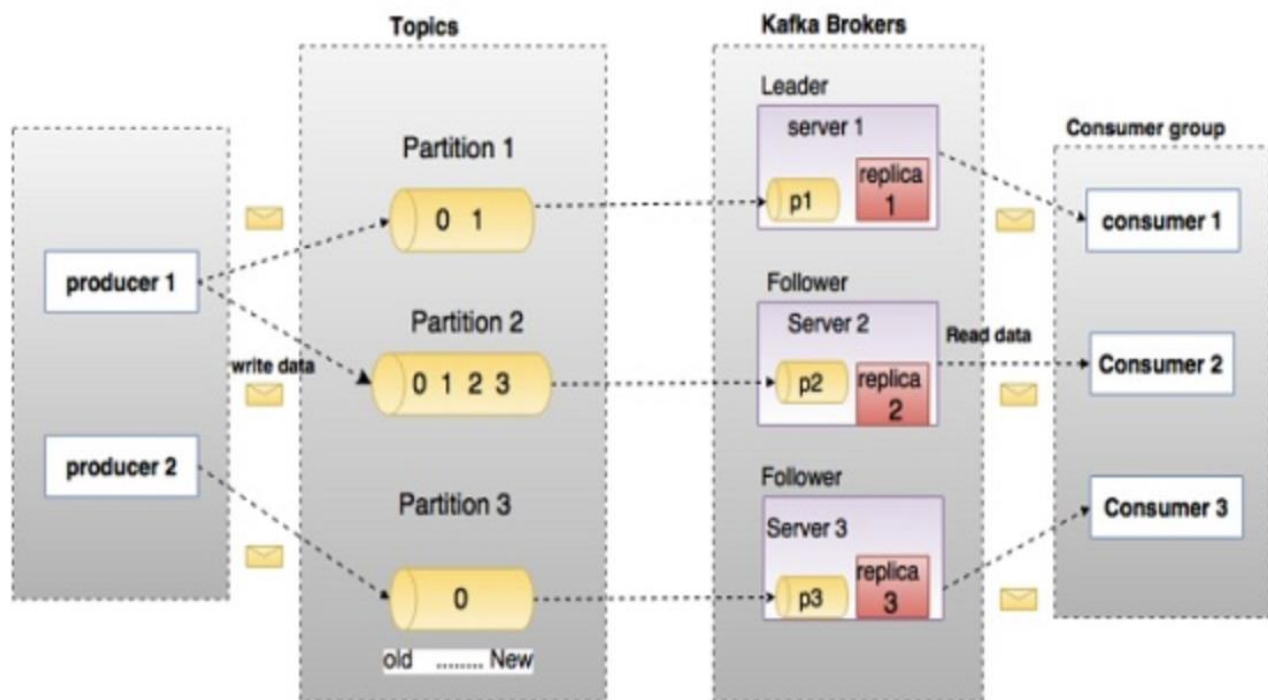


figure 8 :Kafka

In the above diagram, a topic is configured into three partitions. Partition 1 has two offset factors 0 and 1. Partition 2 has four offset factors 0, 1, 2, and 3. Partition 3 has one offset factor 0. The id of the replica is same as the id of the server that hosts it.

Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations. To balance a load in cluster, each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

Components and Description :

Topics

A stream of messages belonging to a particular category is called a topic. Data is stored in topics.

Topics are split into partitions. For each topic, Kafka keeps a minimum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.

Partition

Topics may have many partitions, so it can handle an arbitrary amount of data.

Partition offset

Each partitioned message has a unique sequence id called as “offset”.

Replicas of partition

Replicas are nothing but “backups” of a partition. Replicas are never read or write data. They are used to prevent data loss.

Brokers

- Brokers are simple system responsible for maintaining the published data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.^[1]_{SEP}
- Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.^[1]_{SEP}

- Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distribution among the broker.

Kafka Cluster

Kafka's having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.

Producers

Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.

Consumers

Consumers read data from brokers. Consumers subscribe to one or more topics and consume published messages by pulling data from the brokers.

Leader

“Leader” is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.

Follower

Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store.

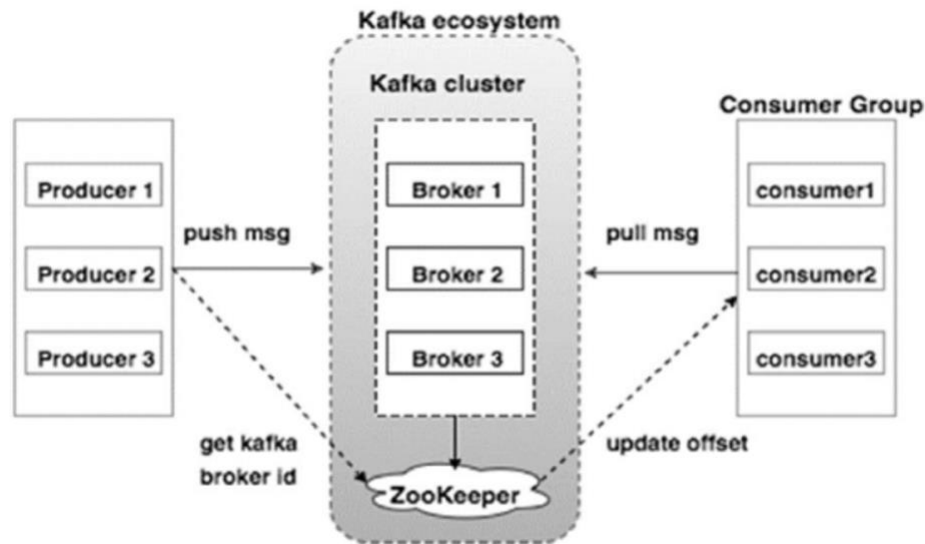


figure 9 : Cluster diagram of kafka

Broker

Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact. Kafka broker leader election can be done by ZooKeeper.

ZooKeeper

ZooKeeper is used for managing and coordinating Kafka broker. ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received

by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.

Producers

Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

Consumers

Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.

Workflow of Pub-Sub Messaging :

Following is the step wise workflow of the Pub-Sub Messaging –

- Producers send message to a topic at regular intervals.
- Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions. If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
- Consumer subscribes to a specific topic.
- Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
- Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.

- Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- Consumer will receive the message and process it.
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outages.
- This above flow will repeat until the consumer stops the request.
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

Workflow of Queue Messaging / Consumer Group

In a queue messaging system instead of a single consumer, a group of consumers having the same “Group ID” will subscribe to a topic. In simple terms, consumers subscribing to a topic with same “Group ID” are considered as a single group and the messages are shared among them. Let us check the actual workflow of this system.

- Producers send message to a topic in a regular interval.
- Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
- A single consumer subscribes to a specific topic, assume “Topic-01” with “Group ID” as “Group-1”.
- Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, “Topic-01” with the same “Group ID” as “Group-1”.
- Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. This sharing will go on until the number of consumers reach the number of partition configured for that particular topic.

- Once the number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribes. This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.
- This feature is also called as “Consumer Group”. In the same way, Kafka will provide the best of both the systems in a very simple and efficient manner.

Role of ZooKeeper

A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service. Zookeeper serves as the coordination interface between the Kafka brokers and consumers. The Kafka servers share information via a Zookeeper cluster. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster. Kafka will restore the state, once the Zookeeper restarts. This gives zero downtime for Kafka. The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.

3.5 Apache Cassandra

Apache Cassandra is highly scalable, high performance, distributed NoSQL database. Cassandra is designed to handle huge amount of data across many commodity servers, providing high availability without a single point of failure.

Cassandra has a distributed architecture which is capable to handle a huge amount of data. Data is placed on different machines with more than one replication factor to attain a high availability without a single point of failure.

NoSQLDatabase

A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

The primary objective of a NoSQL database is to have

- simplicity of design,
- horizontal scaling, and
- finer control over availability.

NoSql databases use different data structures compared to relational databases. It makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it must solve.

Cassandra Architecture

Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Data Replication in Cassandra

In Cassandra, nodes in a cluster act as replicas for a given piece of data. If some of the nodes are responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.

See the following image to understand the schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.

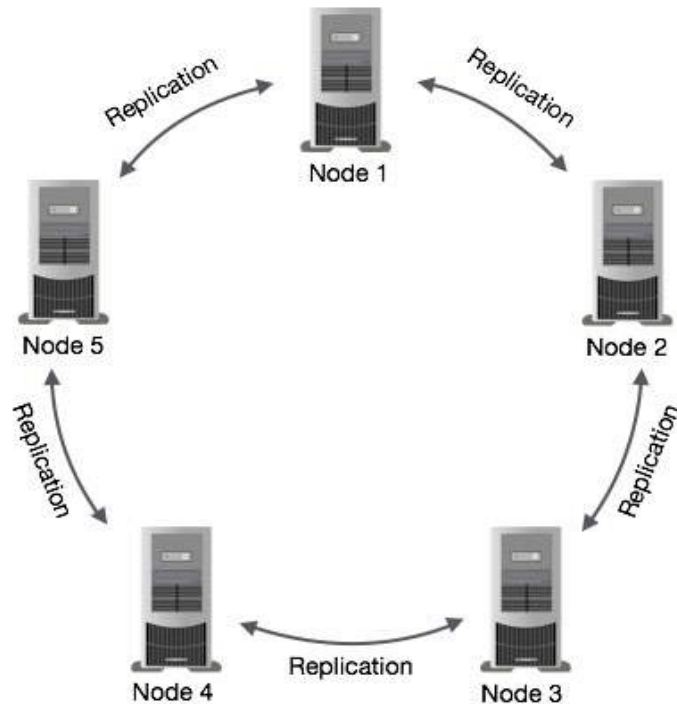


figure 10 : Node Replication

Components of Cassandra :

The main components of Cassandra are:

- Node: A Cassandra node is a place where data is stored.
- Data center: Data center is a collection of related nodes.
- Cluster: A cluster is a component which contains one or more data centers.
- Commit log: In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.

- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Query Language

Cassandra Query Language (CQL) is used to access Cassandra through its nodes. CQL treats the database (Keyspace) as a container of tables. Programmers use `cqlsh`: a prompt to work with CQL or separate application language drivers.

The client can approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

Write Operations

Every write activity of nodes is captured by the commit logs written in the nodes. Later the data will be captured and stored in the mem-table. Whenever the mem-table is full, data will be written into the SSTable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

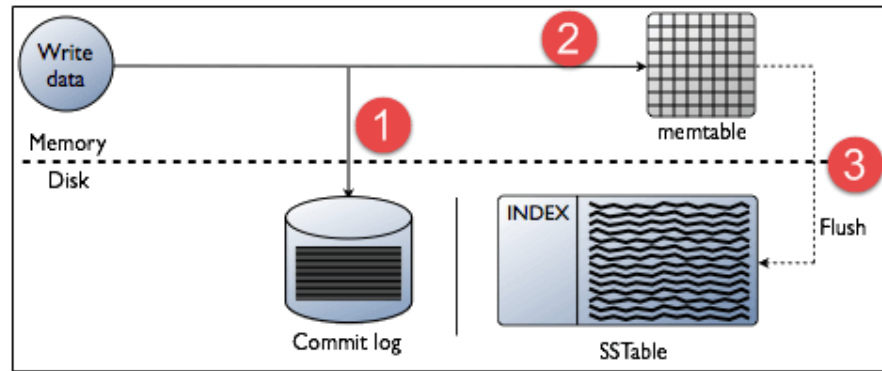


figure 11 : Cassandra Write Operation

Read Operations

In Read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable which contains the required data.

There are three types of read request that is sent to replicas by coordinators.

- Direct request
- Digest request
- Read repair request

The coordinator sends direct request to one of the replicas. After that, the coordinator sends the digest request to the number of replicas specified by the consistency level and checks if the returned data is an updated data.

After that, the coordinator sends digest request to all the remaining replicas. If any node gives out of date value, a background read repair request will update that data. This process is called read repair mechanism.

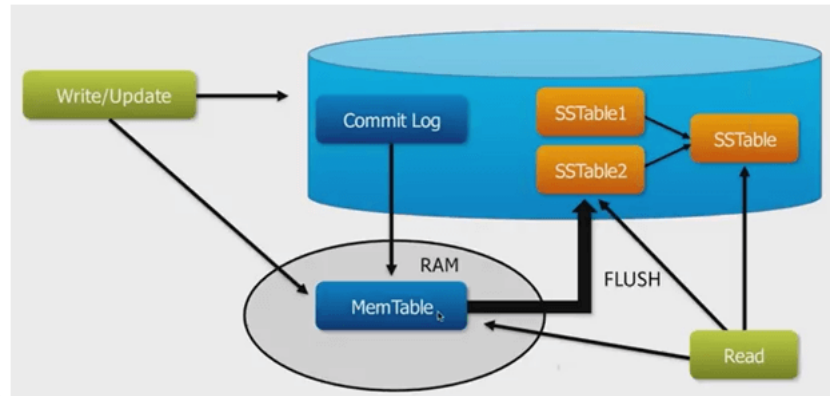


figure 12 : Read Operation

Cassandra Data Model

Data model in Cassandra is totally different from normally we see in RDBMS. Let's see how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that are operated together. The outermost container is known as the Cluster which contains different nodes. Every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. Following are the basic attributes of Keyspace in Cassandra:

- Replication factor: It specifies the number of machine in the cluster that will receive copies of the same data.
- Replica placement Strategy: It is a strategy which species how to place replicas in the ring. There are three types of strategies such as:
 - 1) Simple strategy (rack-aware strategy)
 - 2) old network topology strategy (rack-aware strategy)

3) network topology strategy (datacenter-shared strategy)

- Column families: column families are placed under keyspace. A keyspace is a container for a list of one or more column families while a column family is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

In Cassandra, a well data model is very important because a bad data model can degrade performance, especially when you try to implement the RDBMS concepts on Cassandra.

Cassandra data Models Rules

Cassandra doesn't support JOINS, GROUP BY, OR clause, aggregation etc. So you have to store data in a way that it should be retrieved whenever you want.

Cassandra is optimized for high write performances so you should maximize your writes for better read performance and data availability. There is a tradeoff between data write and data read. So, optimize you data read performance by maximizing the number of data writes. Maximize data duplication because Cassandra is a distributed database and data duplication provides instant availability without a single point of failure.

3.6 MODEL DEVELOPMENT

Flow-chart

Below is the flow chart that will describe the overall flow that Spring follows .

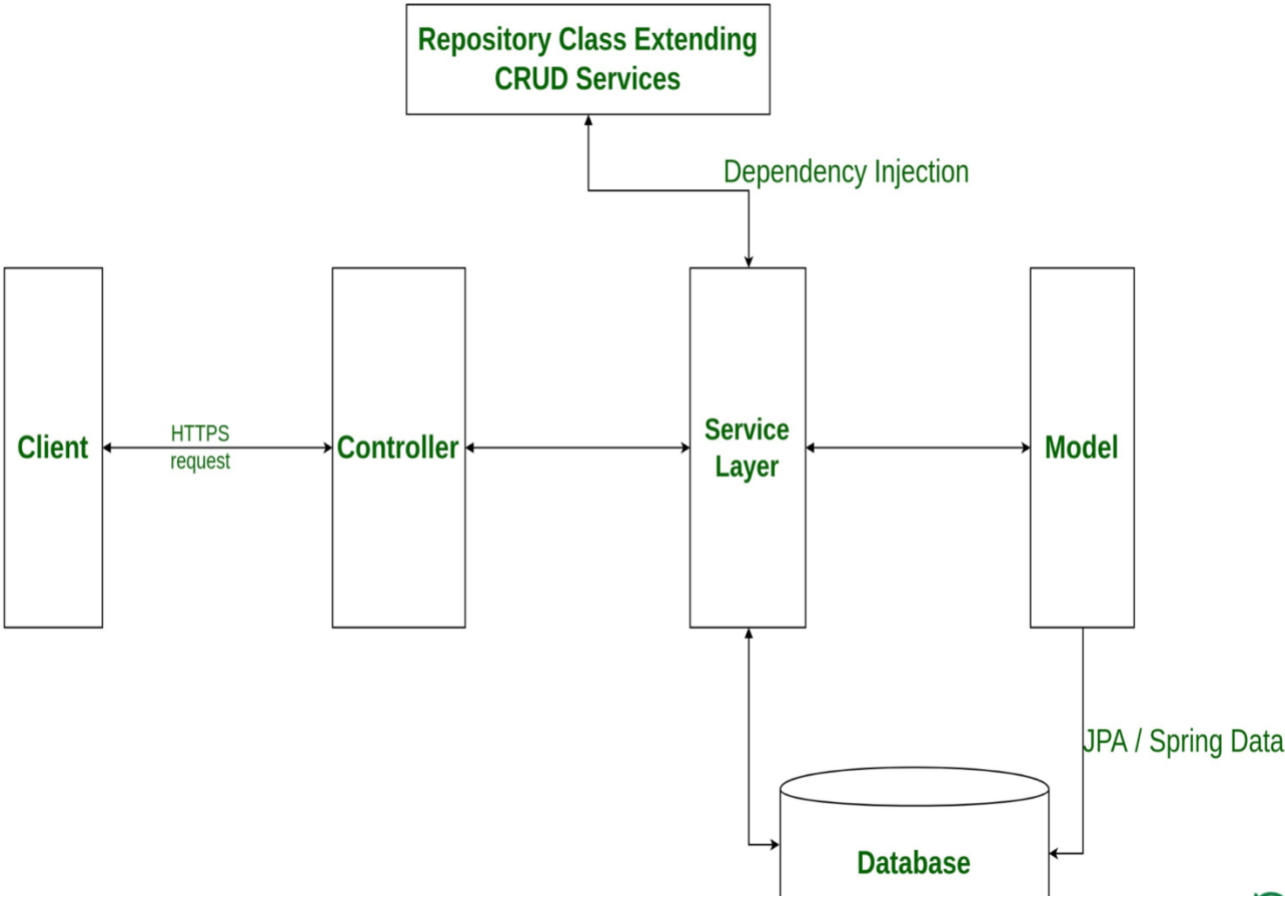


Figure 13 : flow chart

Schema diagram

Below is the Schema diagram of REST that will show what are the services available and who can access them.

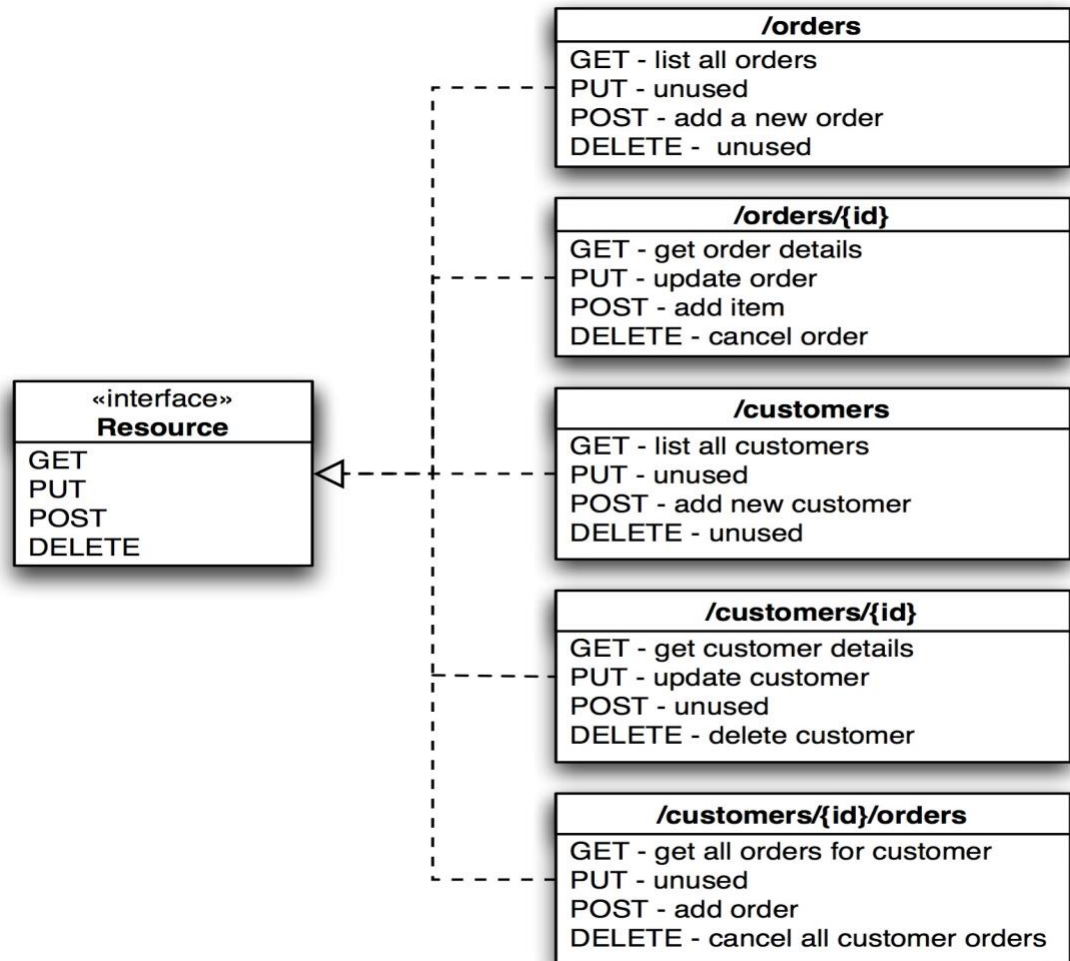


Figure 14 : Schema diagram

3.7 Features

Reactive Programming solves performance problems caused by the use of native threads and the “One thread per request” paradigm. However, this solution goes along with higher development and maintenance complexity because testing and debugging, among other things, become more complicated.

Green threads are a possible way to avoid the performance losses caused by the process switches in the operating system. These were available in Java 1.1 but were already discarded in Java 1.3 because they did not allow the benefits of multi-core or multi-processor systems to be used. A new attempt to introduce another variant of Green Threads, so-called Fibers, into the JDK is Project Loom. This proposal would be accompanied by support for continuations in Java as a kind of spin-off product. This feature is known from other programming languages like Kotlin and Go under the name Coroutines

- avoid “callback hell”
- a lot simpler to do async / threaded work
- a lot of operators that simplify work
- very simple to compose streams of data
- complex threading becomes very easy
- you end up with a more cleaner, readable code base
- easy to implement back-pressure

CHAPTER 4

PERFORMANCE ANALYSIS

4.1 Starting Spring Boot Application

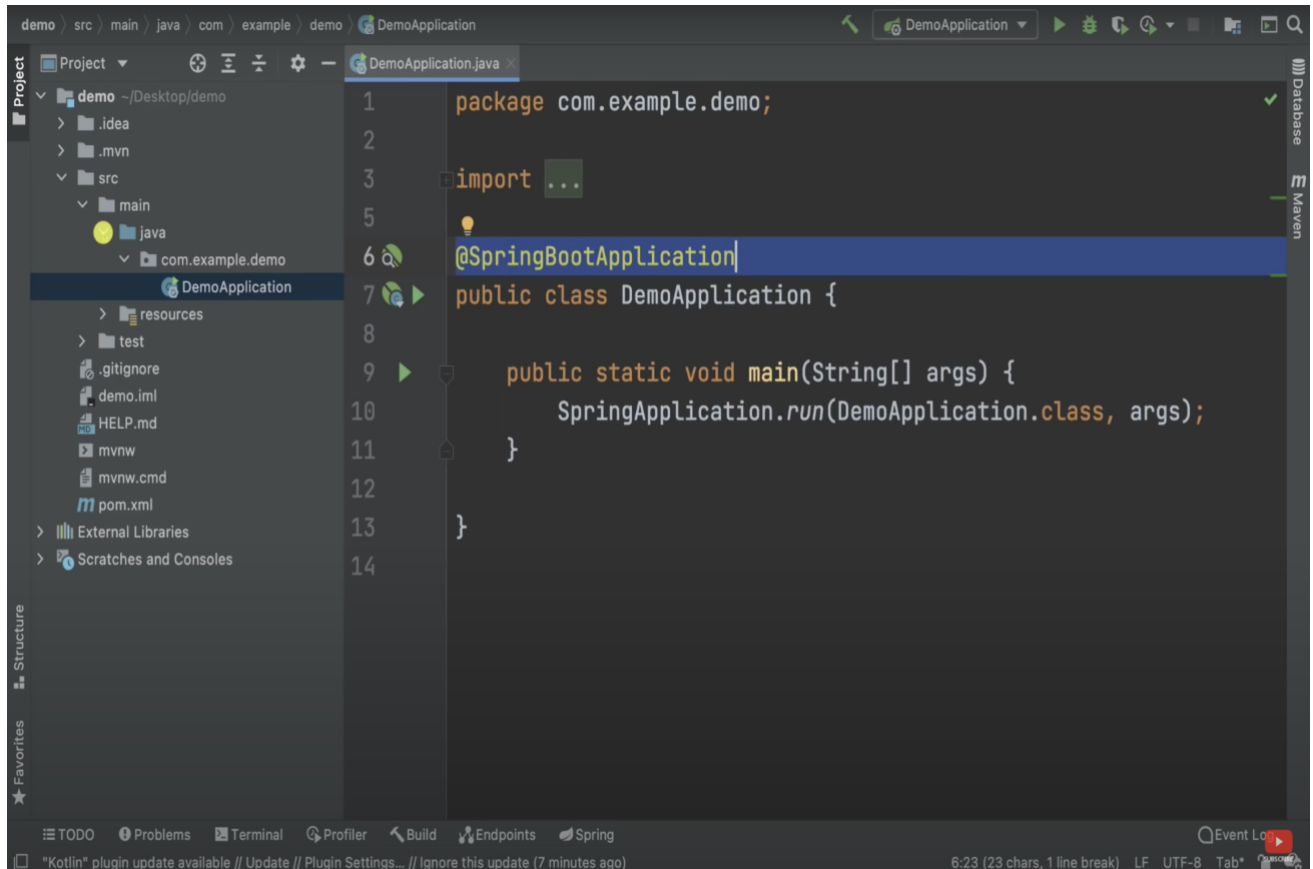
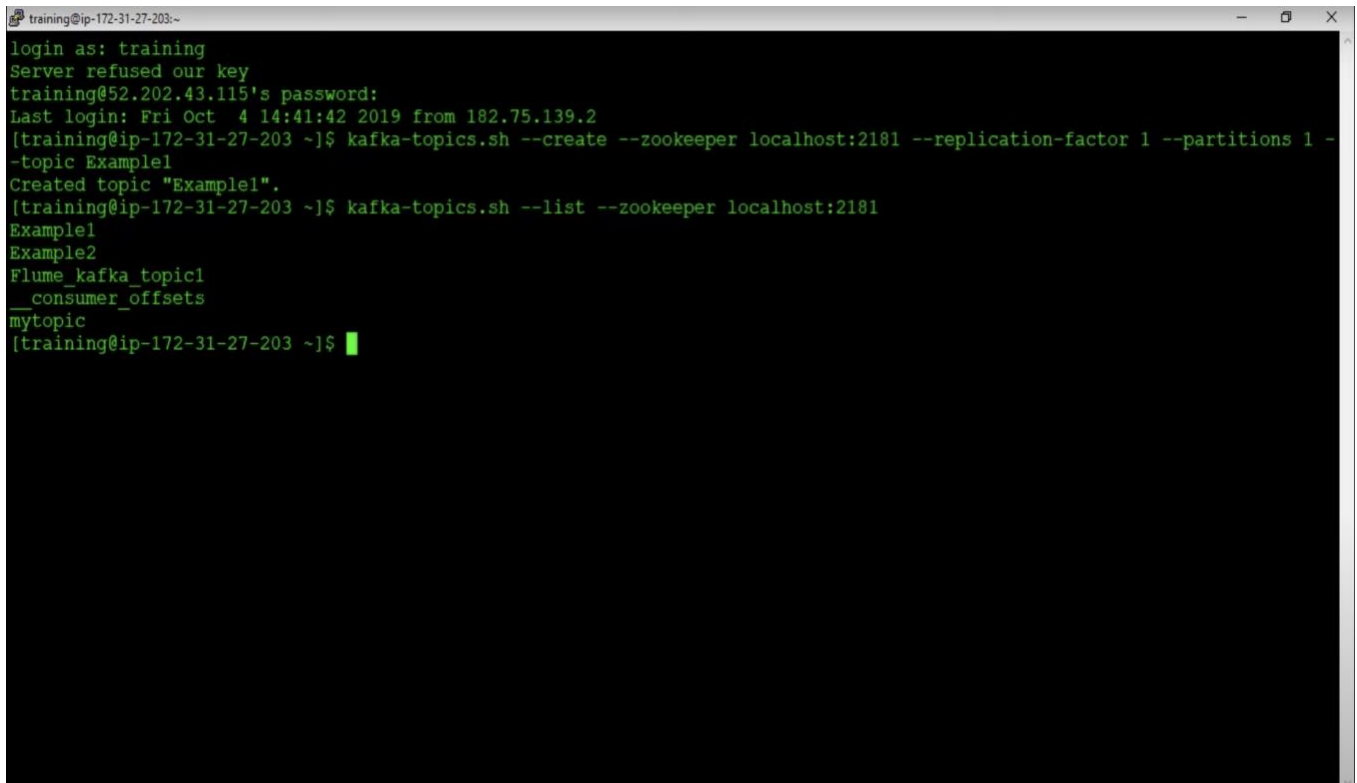


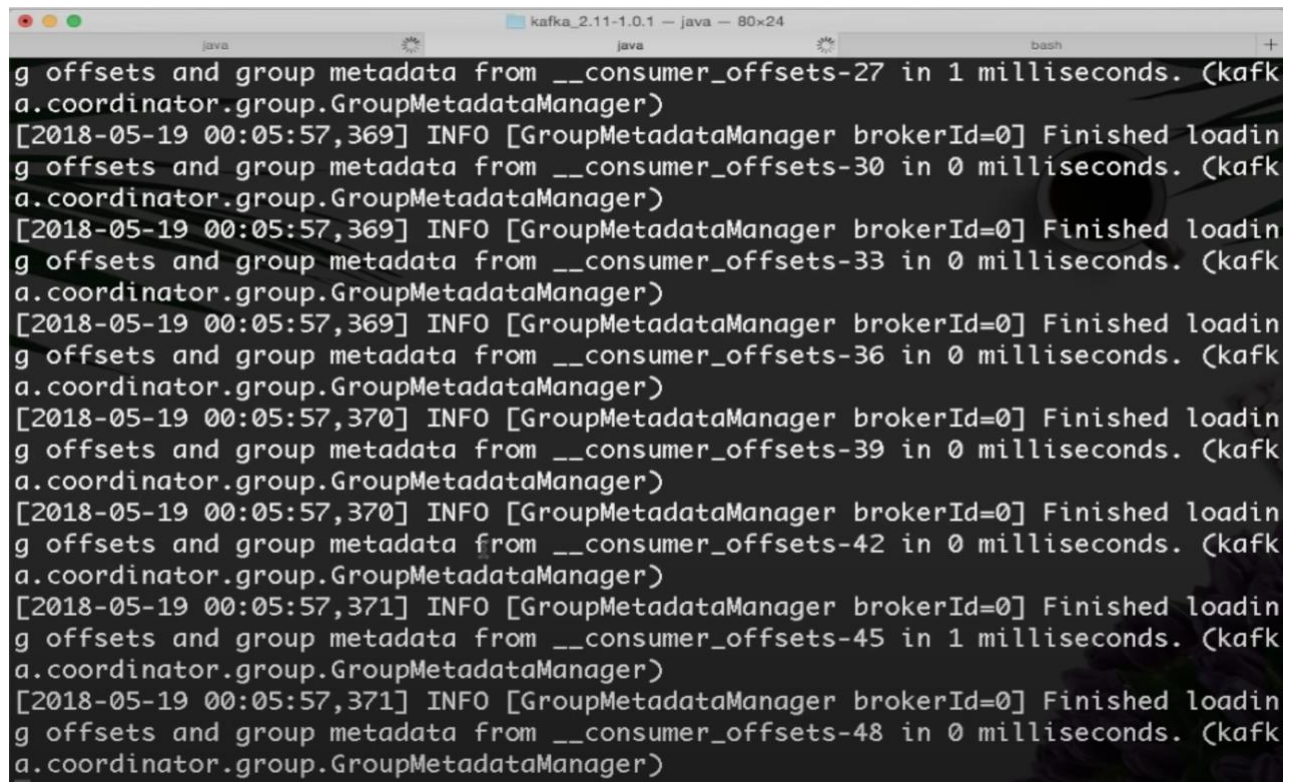
Figure 15: Spring Boot Application

4.2 Kafka



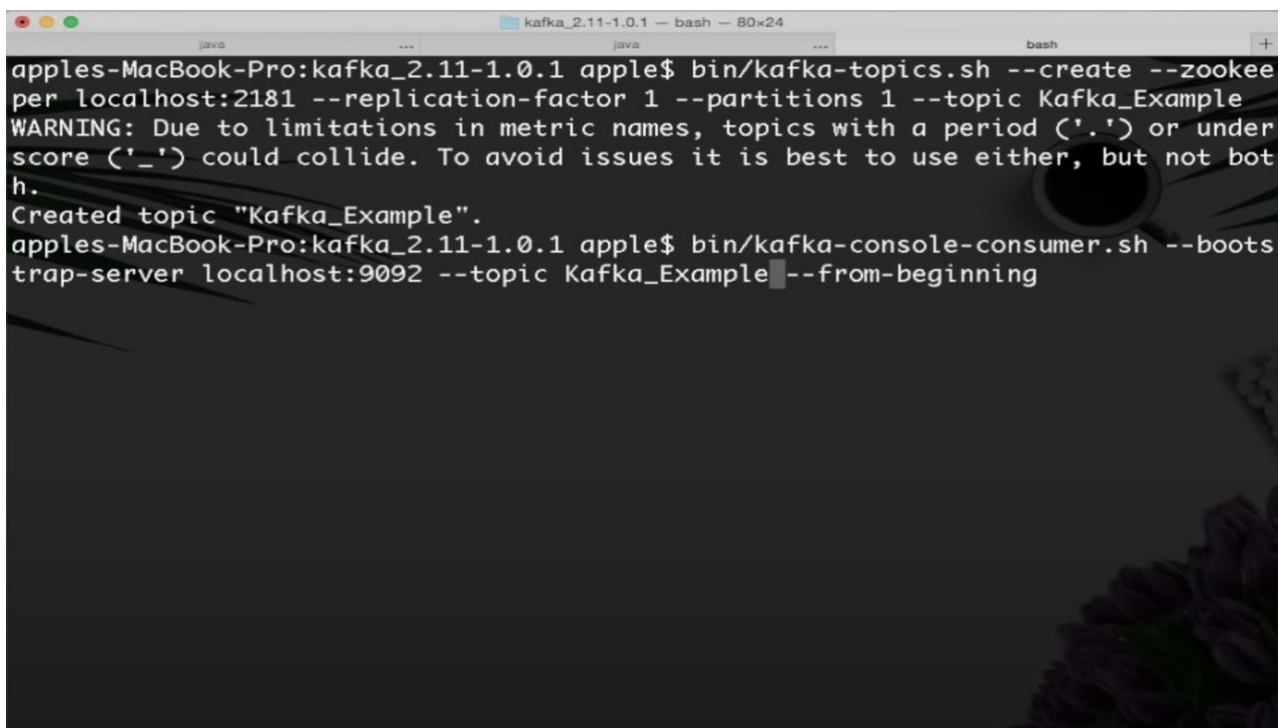
```
training@ip-172-31-27-203:~  
login as: training  
Server refused our key  
training@52.202.43.115's password:  
Last login: Fri Oct  4 14:41:42 2019 from 182.75.139.2  
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Example1  
Created topic "Example1".  
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181  
Example1  
Example2  
Flume_kafka_topic1  
__consumer_offsets  
mytopic  
[training@ip-172-31-27-203 ~]$
```

Figure 16:- Kafka Zookeeper

A screenshot of a terminal window titled 'kafka_2.11-1.0.1 — java — 80x24'. The terminal shows a series of log messages from the Kafka GroupMetadataManager. Each message reports that it has finished loading offsets and group metadata from a specific consumer offset. The offsets shown are 27, 30, 33, 36, 39, 42, 45, and 48. The logs include timestamps like [2018-05-19 00:05:57,369] and [2018-05-19 00:05:57,370], and the log level is INFO. The class name 'kafka.coordinator.group.GroupMetadataManager' is also visible in each log entry.

```
g offsets and group metadata from __consumer_offsets-27 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-30 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-33 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-36 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,370] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-39 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,370] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-42 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,371] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-45 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,371] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-48 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```

Figure 17: Kafka Server

A screenshot of a terminal window titled 'kafka_2.11-1.0.1 — bash — 80x24'. The terminal shows two commands being executed. The first command is 'bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Kafka_Example', which results in a warning about metric names and the creation of the topic 'Kafka_Example'. The second command is 'bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Kafka_Example --from-beginning', which is used to start consuming from the beginning of the 'Kafka_Example' topic.

```
apples-MacBook-Pro:kafka_2.11-1.0.1 apple$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Kafka_Example
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic "Kafka_Example".
apples-MacBook-Pro:kafka_2.11-1.0.1 apple$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Kafka_Example --from-beginning
```

Figure 18: Kafka Consumer

4.3 REST Controller

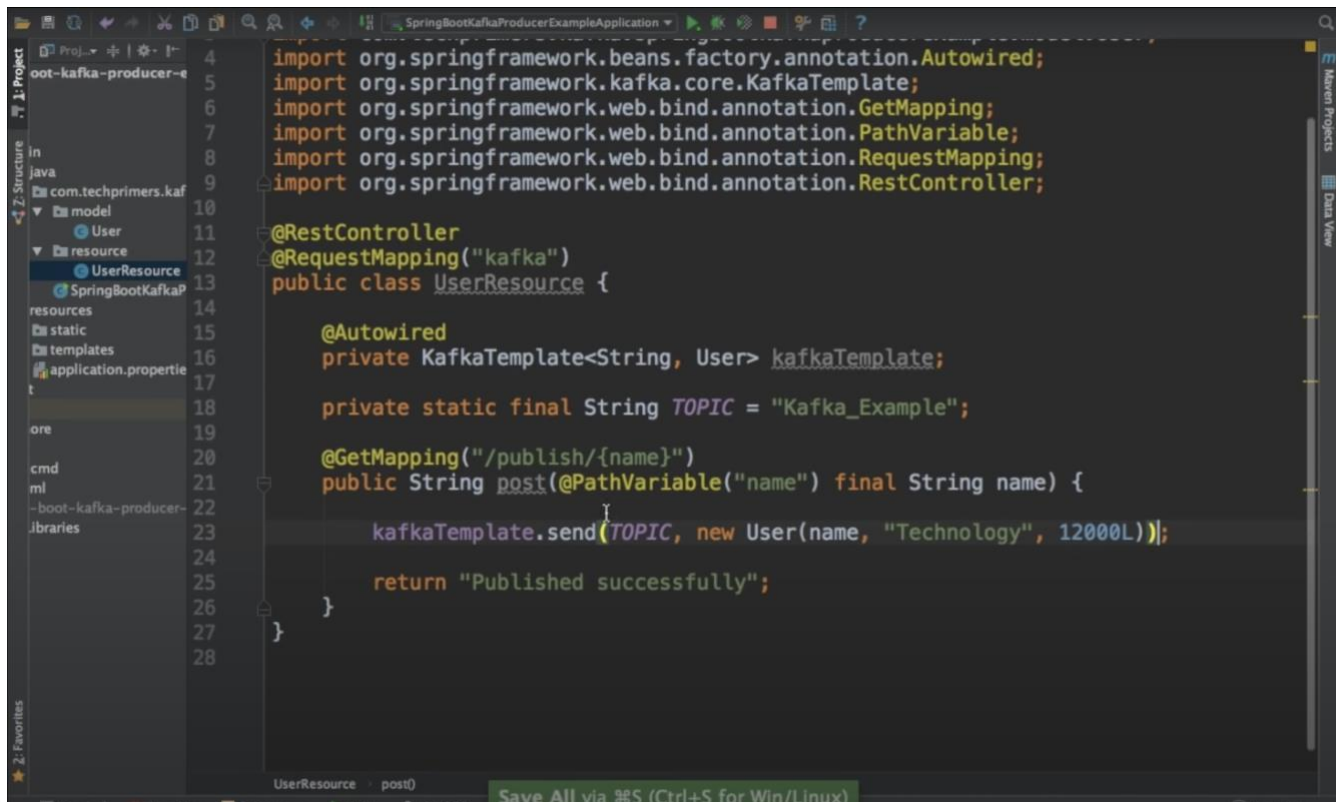


Figure 19: Apache sever Image.


```

C:\Users\User\Desktop\Docker>docker-compose up
Creating network "docker_default" with the default driver
Building web
Step 1/3 : FROM php:7.4-apache
----> 6de63328f8a3
Step 2/3 : COPY site/ /var/www/html/
----> 4b4a1c3100cd
Step 3/3 : EXPOSE 80
----> Running in 593ea199d8ec
Removing intermediate container 593ea199d8ec
----> 3dab95ce19d9

Successfully built 3dab95ce19d9
Successfully tagged docker_web:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Pulling db (mysql:latest)...
latest: Pulling from library/mysql
852e50cd189d: Already exists
29960ddb0fffb: Pull complete
a43f41a44c48: Pull complete
5cdd802543a3: Pull complete
b79b040de953: Pull complete
938c64119969: Pull complete
7689ec51a0d9: Pull complete
a880ba7c411f: Pull complete
984f656ec6ca: Pull complete
9f497bce458a: Pull complete
b9940f97694b: Pull complete
2f069358dc96: Pull complete
Digest: sha256:4bb2e81a40e9d0d59bd8e3dc2ba5e1f2197696f6de39a91e90798dd27299b093
Status: Downloaded newer image for mysql:latest
Creating docker_web_1 ...
Creating docker_web_1 ... error
WARNING: Host is already in use by another container

ERROR: for docker_web_1 Cannot start service web: driver failed programming external connectivity on endpoint docker_web_1 (5b8dec2c56e50b81039f293d5b66ede472c7d55c086
Creating docker_db_1 ... done

ERROR: for web Cannot start service web: driver failed programming external connectivity on endpoint docker_web_1 (5b8dec2c56e50b81039f293d5b66ede472c7d55c086ff0934621
25c4d9de4e81): Bind for 0.0.0.0:80 failed: port is already allocated
ERROR: Encountered errors while bringing up the project.

C:\Users\User\Desktop\Docker>

```

Figure 22: Database Image

```

C:\Users\User\Desktop\Docker>docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
8c5145957d74   mysql     "docker-entrypoint.s..." 2 minutes ago  Up 2 minutes  0.0.0.0:3306->3306/tcp, 33060/tcp  docker_db_1
b1c09c0396fc   php_apache "docker-php-entrypoi..." 6 minutes ago  Up 6 minutes  0.0.0.0:80->80/tcp                php_con

C:\Users\User\Desktop\Docker>docker exec -it 8c5145957d74 bash
root@8c5145957d74:/# mysql -uroot -p12345
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
-> ^C
mysql> show databases;
+-----+
| Database |
+-----+
| foodorder |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql>

```

Figure 23: Database runtime

CONCLUSION

When it comes to huge volumes of data or multi-userness, we often need asynchronous processing to make our systems fast and responsive. In Java, a representative of old object-oriented programming, asynchronicity can become really troublesome and make the code hard to understand and maintain. So, reactive programming is especially beneficial for this ‘purely’ object-oriented environment as it simplifies dealing with asynchronous flows. With its latest releases (starting with Java 8), Java itself has made some attempts to introduce built-in reactivity, yet these attempts are not very popular with developers to date. But there’re some live and regularly updated third-party implementations for reactive programming in Java that help to save the day and thus are particularly loved and cherished by Java developers.

FUTURE WORK

In future we’ll try and create the appliance serverless. The serverless computing model permits you to make and run applications and services while not having to concern infrastructure or servers. It eliminates infrastructure management tasks like server provisioning, patching, software system maintenance, scaling, and capability provisioning. Building native serverless applications implies that developers will specialise in the core product and on innovating applications and solutions, instead of outlay tons of your time on putting in and maintaining infrastructure.

REFERENCE

- [1] A survey on Reactive Programming: Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx and Wolfgang De Meuter
https://www.researchgate.net/publication/233755674_A_Survey_on_Reactive_Programming
- [2] Shantashree Das, Debomalya Ghose, Influence Of Fast Online Apps On The Operations Of The Restaurant Business : INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 8, ISSUE 12, DECEMBER 2019 ISSN 2277-8616
- [3] P.Nagendra Babu , M.Chaitanya Kumari , S.Venkat Mohan [3] has worked on computation of cloud computing, International Journal of Engineering Trends and Technology (IJETT)
– Volume 21 Number 6 – March 2015 ISSN: 2231-5381
- [4] Babak Bashari Rad, Harrison John Bhatti and Mohammad Ahmadi [4] worked on Docker which provide some facilities, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 07 | July 2020

ZopSmart_Technology.pdf

by

Submission date: 24-May-2021 06:12PM (UTC+0530)

Submission ID: 1593112384

File name: ZopSmart_Technology.pdf (2.2M)

Word count: 10188

Character count: 68709

Reactive Spring Boot Application For Product Variant Using Kafka And Cassandra Database

Project report ¹⁰ submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

In

Computer Science and Engineering

By:

Aditya Kumar Singh (171289)

Under the supervision

of

Mr. Aman Sinha

(Tech Lead, ZopSmart Technology)

¹¹
To



Department of Computer Science & Engineering and Information Technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT, SOLAN – 173234, HIMACHAL PRADESH

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled “Reactive Spring Boot Application For Product Variant Using Kafka And Cassandra Database” in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science submitted in the Department of Computer Science Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from February 2021 To May 2021 under the supervision of Mr. Aman Sinha.

The matter embodied in the report has not been appeased for the award of any other degree or diploma.



Aditya Kumar Singh (171289)

This is to verify that the above statement made by candidates is true to the best of my knowledge.



Mr. Aman Sinha

Tech Lead

ZopSmart Technology

Dated:- 22-05-2021

ACKNOWLEDGEMENT

We have taken efforts to do this project. We wish to express our sincere gratitude to Mr. Aman Sinha, Tech Lead, ZopSmart Technology for constantly monitoring and guiding me to the right path in terms of the project. She constantly helped us in our research and the project wouldn't be possible without his constant support.

Secondly, I would also like to thank Lab assistant who helped me a lot in finalizing this project within the limited time frame.

LIST OF CONTENTS

CHAPTER-1	14	INTRODUCTION	1
1.1		Introduction	2
1.2		Problem statement	3
1.3		Objectives	5
1.4		Methodology	5
1.5		Organization:	7
CHAPTER-2		LITERATURE SURVEY	8
ChAPTER-3		SYSTEM DEVELOPMENT	12
3.1		Reactive Programming	14
		• Basics	11
		• Reactive Systems	15
		• Reactive Stream Specification	15
3.2		Spring Boot	17
		• Experimental setup	18
3.3		Rest Api	20
3.4		Apache Kafka	22
3.5		Apache Cassandra	32
3.6		Model Development	39
3.6		Features	41
CHAPTER-4		PERFORMANCE ANALYSIS	42
CONCLUSION			48
FUTURE WORK			48
REFERENCE			49

LIST OF FIGURES

Figure 1	Cloud Computing Services.....	12
Figure 2	Docker architecture	13
Figure 3	Reactive Event Stream.....	14
Figure 4	Spring Initializer	19
Figure 5	URI.....	21
Figure 6	Point to Point Messaging System	24
Figure 7	Publish Subscribe Messaging System.....	24
Figure 8	Kafka	26
Figure 9	Cluster Diagram of Kafka	29
Figure 10	Node Replication	34
Figure 11	Cassandra Write Operation	36
Figure 12	Read Operation	37
Figure 13	Flow Chart	39
Figure 14	Schema diagram	40
Figure 15	Spring Boot Application	42
Figure 16	Kafka ZooKeeper	43
Figure 17	Kafka Server.....	44
Figure 18	Kafka Consumer	44
Figure 19	Apache Server Image.....	45
Figure 20	Docker Image.....	46
Figure 21	Building container	46
Figure 22	Database Image	47
Figure 23	Database runtime	47

Abstract

The Zopsmart Smart Store is a platform to help businesses setup their own customized e-commerce website. It has two variants: Smart Store Eazy, and Smart Store Enterprise Edition. As the name suggests for both Smart Store Enterprise Edition contains many features that make it modular and scalable, it has minimal integration, you can operate it for multiple stores, any many more customer friendly features. Whereas Smart Store Eazy is a lighter version of the Enterprise Edition and is intended for small grocery stores. So, when a client uses it to build his/her own e-commerce platform, it must be ensured that they get access to the features they need, for that multiple extensions have been provided so that the client can curate their website as per their own requirement. When it comes to clients from abroad, or any business that has to extend abroad, there arises the need of multi-lingual support on the website. This helps the customers search in their respective languages and receive the search results in their respective language. But nothing is so easy when it comes to integrate this feature. A separate technology needs to be added in order to handle this and in order to avoid uncontrolled calls to the database. Usage of other technologies (such as Google APIs) helps in translation of the content as per the language variation.

CHAPTER 1

INTRODUCTION

³ Product variants are used to manage products having different variations, like size, colour, etc. It allows managing the product at the template level (for all variations) and at the variant level (specific attributes). As an example, a company selling t-shirts may have the following product: Levi's T-shirt .It will have Sizes: S, M, L, XL, XXL , Colours: Blue, Red, White, Black etc.

In this example, Levi's T-Shirt is called the product template and T-Shirt, S, Blue is a variant. Sizes and color are attributes.

The above example has a total of 20 different products (5 sizes x 4 colors). Each one of these products has its own inventory, sales, etc. An E-commerce website requires a multilingual support because it eases the usage and searching of products for customers using the platform. Also, it as an important aspect if your business is purely internet based. An intuitive user interface is a hallmark of any decent business. But in order for the frontend of applications to work smoothly, you must also consider the backend. Backend development, also called server-side development, handles the behind-the-scenes functions of web development – things like interactions with databases, authorizing users and routing URLs.

Reactive Programming (RP) is a programming model that is designed to cope with asynchronous events (data streams) and the specific act of producing a change, in other words, it means that modifications are implemented to the execution environment in an effective ceratin order. Take a look at the sequences of events in real life in order to have a full understanding of the reactive programming Java paradigm.Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

Cassandra is a distributed database management system designed for handling a high volume of structured data across commodity servers. Cassandra handles the huge amount of data with its distributed architecture. Data is placed on different machines with more than one replication factor that provides high availability and no single point of failure.

2

There are lots of great reasons why you should use reactive programming as a business or developer.

Here are the major ones to think about.

1. Improves user experience - this is at the very heart of why you should be using reactive programming for your apps or websites. The asynchronous nature of FRP means that whatever you program with it will offer a smoother, more responsive product for your users to interact with.

2. Easy to manage - one big bonus with reactive programming is that it is easy to manage as a developer. Blocks of code can be added or removed from individual data streams which means you can easily make any amendments needed via the stream concerned.

3. Simpler than regular threading - FRP is less hassle than regular threading due to the way it allows you to work on the data streams. Not only is this true for basic threading

in an application but also for more complex threading operations you may need to undertake.

Problem statement

- **Thread Per Request Model:-**

The application will only be able to handle a number of concurrent requests that equals the size of the thread pool. It is possible to configure the size of the thread pool, but since each thread reserves some memory (typically 1MB), the higher thread pool size we configure, the higher the memory consumption. If the application is designed according to a microservice based architecture, we have better possibilities to scale based on load, but a high memory utilization still comes with a cost. that the greatest advantage of cloud computing. It encourages ¹ you to spare important capital expense because it needn't trouble with any actual Instrumentality ventures.

- **Waiting for I/O operation:-**

Same type of waste also occurs while waiting for other types of I/O operations to complete such as a database call or reading from a file. In all these situations the thread making the I/O request will be blocked and waiting idle until the I/O operation has completed, this is called blocking I/O. Such situations where the executing thread gets blocked, just waiting for a response, means a waste of threads and therefore a waste of memory.

- **Response Time:-**

Another issue with traditional imperative programming is the resulting response times when a service needs to do more than one I/O request. For example, service A might need to call service B and C as well as do a database lookup and then return some aggregated data as a result. This would mean that service A's response time would besides its own processing time be a sum of:

- response time of service B (network latency + processing)
- response time of service C (network latency + processing)
- response time of database request (network latency + processing)

- **Overwhelming the client:-**

Another type of problem that might occur in a microservice landscape is when service

A is requesting some information from service B, let's say for example all the orders placed during last month. If the amount of orders turns out to be huge, it might become a problem for service A to retrieve all this information at once. Service A might be overwhelmed with the high amount of data and it might result in for example an out of memory-error.

- **On Demand Self Service**

Developer not have to be compelled to worry regarding the resources. Resources square measure created on the market to the user on AN “as needed” basis. instead of all quickly , on-demand computing permits cloud hosting firms to supply their purchasers with access to computing resources as they become necessary

The different issues described above are the issues that reactive programming is intended to solve. In short, the advantages that comes with reactive programming is that we:

- move away from the thread per request model and can handle more requests with a low number of threads.
- prevent threads from blocking while waiting for I/O operations to complete.
- make it easy to do parallel calls.
- support “back pressure”, giving the client a possibility to inform the server on how much load it can handle.

Needs of Microservices

1. Continuous Delivery

Microservices provide the ideal architecture for continuous delivery. With microservices, each application resides in a separate container along with the environment it needs to run. Because of this, each application can be edited in its container without the risk of interfering with any other application.

This means zero downtime for users, simplified troubleshooting, and no disruption even if a problem is identified. The safe and rapid changes allowed by microservice architecture make it possible to update software fast enough to put the “continuous” in continuous delivery. By keeping disruption to a minimum, microservice architecture lets you update rapidly without inconveniencing customers.

2. Maximize Deployment Activity

Microservice architecture allows you to maximize deployment velocity and

application reliability by helping you move at the speed of the market. Since applications each run in their own containerized environment, applications can be moved anywhere without altering the environment. If an application works in development, it will work for the customer. This speeds up time to market and increases product reliability

3. Faster innovation to adapt to changing market conditions

Microservices can also help you adapt more quickly to the changing market conditions. Because microservices allow applications to be updated and tested quickly, you can follow market trends and adapt your products faster.

Microservices also give you an edge when it comes to innovation, since developers can experiment on applications without fear of causing problems elsewhere. In today's rapidly changing market, getting an edge on innovation helps you maintain your current revenue streams while driving new revenue

Objectives

To make a Spring Boot non-blocking applications that are asynchronous and event-driven and require a small number of threads to scale. A key aspect of that definition is the concept of backpressure which is a mechanism to ensure producers don't overwhelm consumers .

Methodology

It is a system of broad principles or rules from which specific methods or procedures may be derived to interpret or solve different problems within the scope of a particular discipline. Unlike an algorithm, a methodology is not any formula but it is a set of practices.

- **Amazon Web Services**

Services that can be used are:-

- Amazon Elastic Compute Cloud (Amazon EC2) to run Linux or Windows based servers
- Elastic Load Balancing (ELB) to load balance and distribute the web traffic
- Amazon Elastic Block Store (Amazon EBS) or Amazon Elastic File System (Amazon EFS) to store static content.
- Amazon Virtual Private Cloud (Amazon VPC) to deploy Amazon EC2 instances. Amazon VPC is your isolated and private virtual network in the AWS Cloud and gives you full control over the network topology, firewall configuration, and routing rules.
- Web servers can be spread across multiple Availability Zones for high availability, even if an entire data center were to be down.
- AWS Auto Scaling automatically adds servers during high traffic periods and scales back when traffic decreases

- **Docker**

Docker may be a instrumentation platform that packages your application and every one its dependencies along within the sort of a dockhand container to confirm that your application works seamlessly in any atmosphere. dockhand may be a powerful tool for making and deploying applications. It simplifies rolling out applications across multiple systems and may be a useful gizmo for desegregation new technologies. Associate in Nursing application that runs victimisation dock- hand can start off identical each time occasion anytime on every system. this suggests that if the applying works on your native laptop, it'll work anyplace that supports dockhand.

- **Docker Image**

A dockhand image may be a read-only example that contains a collection of Direc- tions for making a instrumentality that may run on the dockhand platform. It prov- ides a convenient thanks to package up applications and preconfigured server environments, that you'll be able to use for your own non- public use or share pub- lically with different dockhand users.

- **Docker Containers**

Containers are the style of software virtualization. one instrumentality may well be accustomed run something from a tiny low ⁷ micro-service or code method to a bigger application. within a instrumentality are all the mandatory executables, computer code, libraries, and configuration files. Compared to server or machine virtualization approaches, however, containers don't contain software pictures. This makes them a lot of light-weight and moveable, with considerably ¹⁵ less overhead. In larger application deployments, multiple instrumentality is also ¹⁵ deployed collectively or a lot of container clusters. Such clusters may well be managed by a instrumentality adapter like Kubernetes.

1.5 Organization

This report has been organized with the following chapters:

Chapter 2: In this chapter the previous end related work done in the development of this project were described with their methodology and architecture proposed by the authors.

Chapter 3: In this chapter we see how we work on system design. Also described Spring Boot , Apache Kafka , Reactive Programming , REST and Cassandra Database which will be used in the development of spring boot application.

Chapter 4: Discusses about the result and Screenshots.

Chapter 5: Concludes the project and gives suggestions for future work.

CHAPTER 2

LITERATURE SURVEY

[1] Dr. Mitali Gupta [1] conducted the study of Reactive app that are available in India and concluded that only the application that were successful have good user interface and shows the image in a way that sticks in the mind of user so he or she will buy that. In short a good user interface application and low latency application gives best user experience .

[2] Shantashree Das, Debomalya Ghose [2] conducted study on influence of online food delivery application on the operation on business the paper discuss about how human behaviour are changing and they are more inclined towards ordering food online instead of going to restaurant and also provided some solution in order to operate business in more efficient manner. So Reactive Spring boot is making better user experience

[3] P.Nagendra Babu , M.Chaitanya Kumari , S.Venkat Mohan [3] has worked on computation of cloud computing , data access and storage services that do not require end user knowledge of the physical location and configuration of the system that delivers the services.

SAAS	PAAS	IAAS
Software As A Service	Platform As A Service	Infrastructure As A Service
<ul style="list-style-type: none"> ✓ Government Applications ✓ Communications ✓ Productivity tools 	<ul style="list-style-type: none"> ✓ Application Development ✓ Security Services ✓ Database Management 	<ul style="list-style-type: none"> ✓ Server ✓ Network ✓ Storage
Examples: <ul style="list-style-type: none"> ✓ Oracle ✓ SalesForce.com ✓ LinkedIn ✓ Google Apps 	Examples: <ul style="list-style-type: none"> ✓ Microsoft Azure ✓ GAE 	Examples: <ul style="list-style-type: none"> Amazon EC2 Verizon Terre mark

Figure 1 Cloud Computing Services

Concluded that however Cloud is employed for organizations and the way the cloud is employed to Store , retrieve and modify the info while not physical instrumentation .Cloud computing is that the quickest new paradigm for delivering on demand services over web and might be represented as central Software system.Cloud computing describes a brand new supplement , consumption and delivery model for IT services supported web protocols and it generally involves provisioning of dynamically ascendable and infrequently virtualized resources .

[4] BabakBashari Rad, Harrison John Bhatti and Mohammad Ahmadi [4] worked on Docker which provide some facilities, which are useful for developers and administrators. It is an open platform can be used for building ,distributing, and running applications in a portable , lightweight runtime and packaging tool, known as Docker Engine. It also provide Docker Hub, which is a cloud service for sharing applications. Costs can be reduced by replacing traditional virtual machine with docker container. It excellently reduces the cost of re-building the cloud development platform.

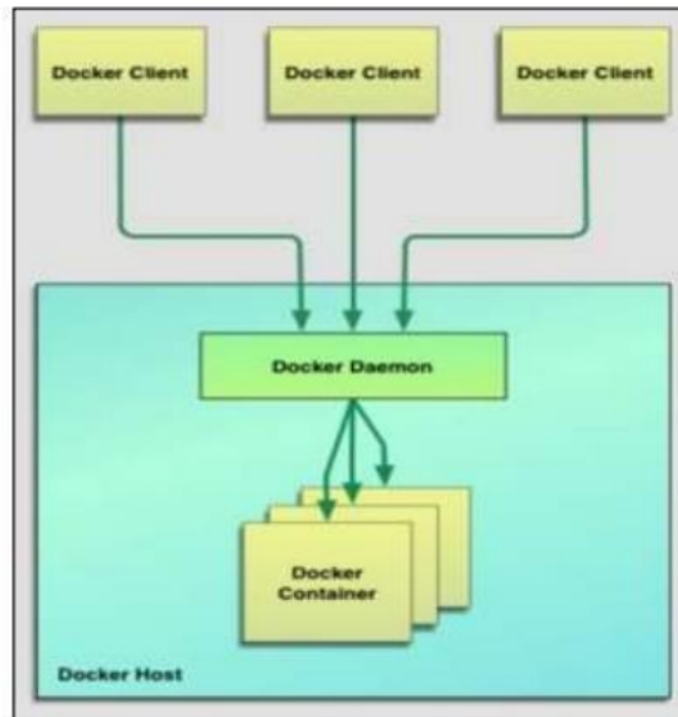


Figure 2 Docker architecture

The conclusion is Docker automates the applications when they are containerized. An extra layer of docker engine is added to the host operating system. The performance of docker is faster than virtual machines as it has no guest operating system and less resourceoverhead.

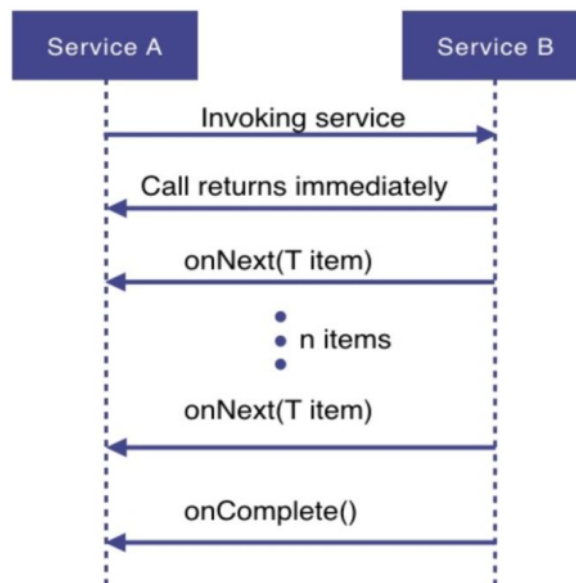
CHAPTER 3

SYSTEM DEVELOPMENT

3.1 Reactive Programming

- Basics

In short: by programming with asynchronous data streams. Let's say service A wants to retrieve some data from service B. With the reactive programming style approach, service A will make a request to service B which returns immediately (being non-blocking and asynchronous). Then the data requested will be made available to service A as a data stream, where service B will publish an `onNext`-event for each data item one by one. When all the data has been published, this is signalled with an `onComplete` event. In case of an error, an `onError` event would be published and no more items would be emitted.



1

figure 3 : Reactive Event Stream

Reactive programming uses a functional style approach (similar to the Streams API), which gives the possibility to perform different kinds of transformations on the streams. A stream can be used as an input to another one. Streams can be merged, mapped and filtered .

- **Reactive Systems**

Reactive programming is an important implementation technique when developing “reactive systems”, which is a concept described in the “Reactive Manifesto”, highlighting the need for modern applications to be designed to be:

1. Responsive (responding in a timely manner)
2. Resilient (staying responsive also in failure situations)
3. Elastic (staying responsive under varying workload)
4. Message Driven (relying on asynchronous message passing)

Building a reactive system means to deal with questions such as separation of concerns, data consistency, failure management, choice of messaging implementation etc. Reactive programming can be used as an implementation technique to ensure that the individual services use an asynchronous, non-blocking model, but to design the system as a whole to be a reactive system requires a design that takes care of all these other aspects as well.

- **Reactive Streams Specification**

As time went on, a standardisation for Java was developed through the Reactive Streams effort. Reactive Streams is a small specification intended to be implemented by the reactive libraries built for the JVM. It specifies the types to implement to achieve interoperability between different implementations. The specification defines the interaction between asynchronous components with back pressure. Reactive Streams was adopted in Java 9, by the Flow API . The purpose of the Flow API is to act as an interoperation specification and not an end-user API like RxJava.

The specification covers the following interfaces:

Publisher : This represents the data producer/data source and has one method which lets the subscriber register to the publisher.

```
public interface Publisher<T> {  
    public void subscribe(Subscriber<? super T> s);  
}
```

Subscriber : This represents the consumer and has the following methods:

```
public interface Subscriber<T> {  
    public void onSubscribe(Subscription s);  
    public void onNext(T t);  
    public void onError(Throwable t);  
    public void onComplete();  
}
```

- onSubscribe is to be called by the Publisher before the processing starts and is used to pass a Subscription object from the Publisher to the Subscriber^[LSEP]
- onNext is used to signal that a new item has been emitted^[LSEP]
- onError is used to signal that the Publisher has encountered a failure and no more items will be emitted
- onComplete is used to signal that all items were emitted successfully^[LSEP]

Subscription : The subscriptions holds methods that enables the client to control the Publisher's emission of items (i.e. providing backpressure support).

```
public interface Subscription {  
    public void request(long n);  
    public void cancel();  
}
```

- request allows the Subscriber to inform the Publisher on how many additional elements to be published

- cancel allows a subscriber to cancel further emission of items by the Publisher.

Processor : If an entity shall transform incoming items and then pass it further to another Subscriber, an implementation of the Processor interface is needed. This acts both as a Subscriber and as a Publisher.

```
public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {
}
```

16

3.2 Spring Boot

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

How does it work ?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using `@EnableAutoConfiguration` annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains `@SpringBootApplication` annotation and the main method.

Spring Boot automatically scans all the components included in the project by using `@ComponentScan` annotation.

Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include `spring-boot-starter-data-jpa` dependency in your project.

Note that all Spring Boot starters follow the same naming pattern `spring-boot-starter-*`, where `*` indicates that it is a type of the `application`.

Auto Configuration

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add `@EnableAutoConfiguration` annotation or `@SpringBootApplication` annotation to your main class file. Then, your Spring Boot application will be automatically configured.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
```

```
@EnableAutoConfiguration
```

```
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Spring Boot Application

The entry point of the Spring Boot Application is the class contains `@SpringBootApplication` annotation. This class should have the main method to run the Spring Boot application. `@SpringBootApplication` annotation includes Auto- Configuration, Component Scan, and Spring Boot Configuration.

If you added `@SpringBootApplication` annotation to the class, you do not need to add the `@EnableAutoConfiguration`, `@ComponentScan` and `@SpringBootConfiguration` annotation. The `@SpringBootApplication` annotation includes all other annotations.

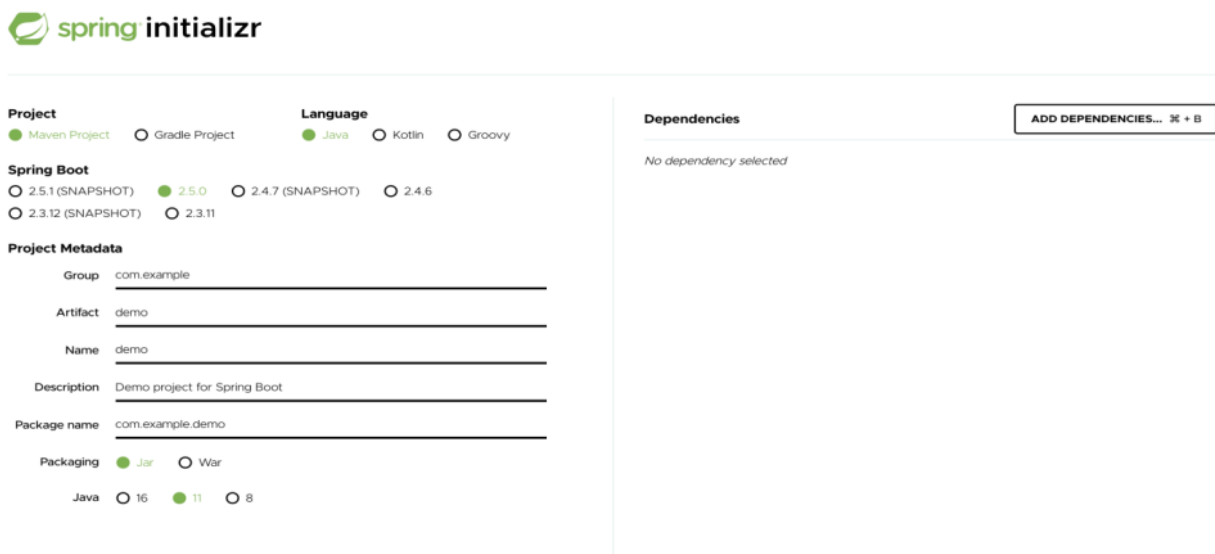
Component Scan

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the `@ComponentScan` annotation for your class file to scan your components added in your project.

Spring Initializer

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page www.start.spring.io and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.

Observe the following screenshot that shows an example where we added the **spring-boot-starter-web** dependency to write REST Endpoints.



The screenshot shows the Spring Initializer web interface. It includes sections for Project, Language, Spring Boot version, Project Metadata, and Dependencies. The Project section has radio buttons for Maven Project (selected) and Gradle Project. The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for various versions, with 2.5.0 selected. The Project Metadata section contains input fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). The Packaging section has radio buttons for Jar (selected) and War. The Java section has radio buttons for 16, 11 (selected), and 8. The Dependencies section has a button 'ADD DEPENDENCIES...' and the text 'No dependency selected'.

Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.5.1 (SNAPSHOT) ☒ 2.5.0 ☐ 2.4.7 (SNAPSHOT) ☐ 2.4.6
☐ 2.3.12 (SNAPSHOT) ☐ 2.3.11

Project Metadata
Group
Artifact
Name
Description
Package name

Packaging
☒ Jar ☐ War

Java
☐ 16 ☒ 11 ☐ 8

Dependencies

No dependency selected

figure 4 : Spring initializer

Properties File

Properties files are used to keep 'N' number of properties in a single file to run the application in a different environment. In Spring Boot, properties are kept in the application.properties file under the classpath.

The application.properties file is located in the src/main/resources directory.

YAML File

Spring Boot supports YAML based properties configurations to run the application. Instead of application.properties, we can use application.yml file. This YAML file also should be kept inside the classpath.

3.3 REST API

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

- **GET** – Provides a read only access to a resource.
- **POST** – Used to create a new resource.
- **DELETE** – Used to remove a resource.
- **PUT** – Used to update a existing resource or create a new ¹resource.

Restful Webservices

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Sr.No.	URI	HTTP Method	POST body	Result
1	/UserService/users	GET	empty	Show list of all the users.
2	/UserService/addUser	POST	JSON String	Add details of new user.
3	/UserService/getUser/:id	GET	empty	Show details of a user.

figure 5 : URI

4

What is a Resource?

REST architecture treats every content as a resource. These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data. REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource where Text, JSON, XML. The most popular representations of resources are XML and JSON.

Representation of Resources

A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database. Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

For example, in Restful Web Services – First Application chapter, a user is a resource which is represented using the following XML format –

```
<user>
  <id>1</id>
  <name>Mahesh</name>
  <profession>Teacher</profession>
</user>
```

The same resource can be represented in JSON format as follows

```
{
  "id":1,
  "name":"Mahesh",
  "profession":"Teacher"
}
```

Good Resources Representation

REST does not impose any restriction on the format of a resource representation. A client can ask for JSON representation whereas another client may ask for XML representation of the same resource to the server and so on. It is the responsibility of the REST server to pass the client the resource in the format that the client understands.

Following are some important points to be considered while designing a representation format of a resource in RESTful Web Services.

- Understandability – Both the Server and the Client should be able to understand and utilize the representation format of the resource.

- Completeness – Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.

3.4 Apache Kafka

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

What is a Messaging System?

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system. Two types of messaging patterns are available – one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow pub-sub.

Point to Point Messaging System

In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.

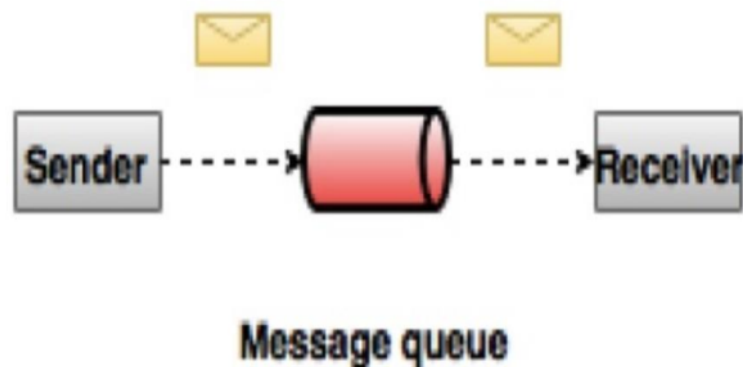


figure 6 : Point to Point Messaging System

Publish-Subscribe Messaging System

In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

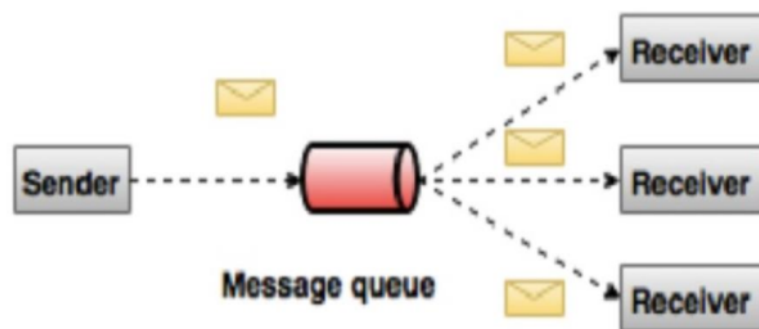


figure 7 : Publish-Subscribe Messaging System

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

Benefits

Following are a few benefits of Kafka –

- Reliability – Kafka is distributed, partitioned, replicated and fault tolerance.
- Scalability – Kafka messaging system scales easily without down time.
- Durability – Kafka uses “Distributed commit log” which means messages persists on disk as fast as possible, hence it is durable.
- Performance – Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

Use Cases

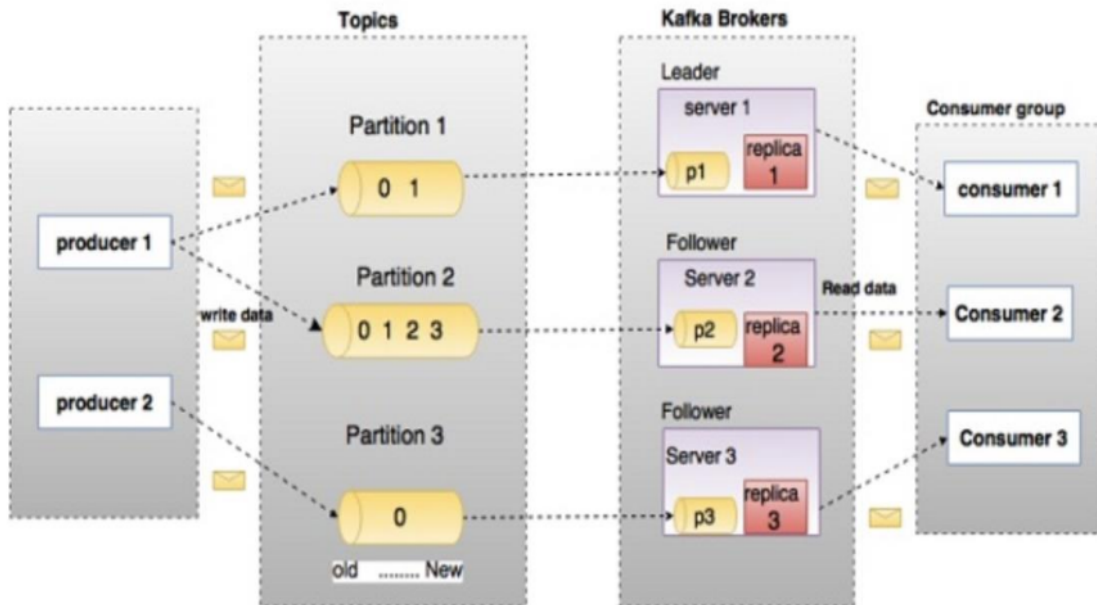
Kafka can be used in many Use Cases. Some of them are listed below –

- Metrics – Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
- Log Aggregation Solution – Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.
- Stream Processing – Popular frameworks such as Storm and Spark Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications. Kafka’s strong durability is also very useful in the context of stream processing.

Need for Kafka

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It has the ability to handle a large number of diverse consumers. Kafka is very fast, performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

Before moving deep into the Kafka, you must aware of the main terminologies such as topics, brokers, producers and consumers. The following diagram illustrates the main terminologies and the table describes the diagram components in detail.



1

figure 8 :Kafka

In the above diagram, a topic is configured into three partitions. Partition 1 has two offset factors 0 and 1. Partition 2 has four offset factors 0, 1, 2, and 3. Partition 3 has one offset factor 0. The id of the replica is same as the id of the server that hosts it.

Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations. To balance a load in cluster, each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

Components and Description :

Topics

A stream of messages belonging to a particular category is called a topic. Data is stored in topics.

Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.

Partition

Topics may have many partitions, so it can handle an arbitrary amount of data.

Partition offset

Each partitioned message has a unique sequence id called as “offset”.

Replicas of partition

Replicas are nothing but “backups” of a partition. Replicas are never read or write data. They are used to prevent data loss.

Brokers

- Brokers are simple system responsible for maintaining the pub-lished data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.^[1]_{SEP}
- Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.^[1]_{SEP}

- Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distribution among the broker.

Kafka Cluster

Kafka's having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.

Producers

Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.

Consumers

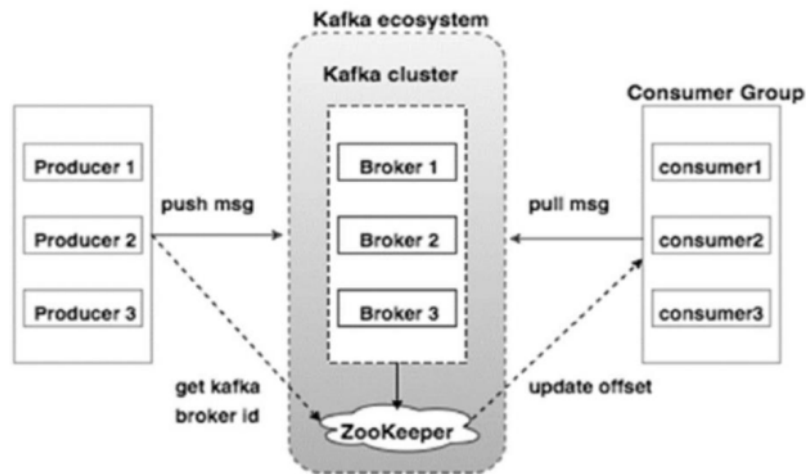
Consumers read data from brokers. Consumers subscribe to one or more topics and consume published messages by pulling data from the brokers.

Leader

"Leader" is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.

Follower

Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store.



1

figure 9 : Cluster diagram of kafka

Broker

Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact. Kafka broker leader election can be done by ZooKeeper.

ZooKeeper

ZooKeeper is used for managing and coordinating Kafka broker. ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received

by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.

Producers

Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

Consumers

Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.

Workflow of Pub-Sub Messaging :

Following is the step wise workflow of the Pub-Sub Messaging –

- Producers send message to a topic at regular intervals.
- Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions. If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
- Consumer subscribes to a specific topic.
- Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
- Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.

- Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- Consumer will receive the message and process it.
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outages.
- This above flow will repeat until the consumer stops the request.
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

Workflow of Queue Messaging / Consumer Group

In a queue messaging system instead of a single consumer, a group of consumers having the same “Group ID” will subscribe to a topic. In simple terms, consumers subscribing to a topic with same “Group ID” are considered as a single group and the messages are shared among them. Let us check the actual workflow of this system.

- Producers send message to a topic in a regular interval.
- Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
- A single consumer subscribes to a specific topic, assume “Topic-01” with “Group ID” as “Group-1”.
- Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, “Topic-01” with the same “Group ID” as “Group-1”.
- Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. This sharing will go on until the number of consumers reach the number of partition configured for that particular topic.

- Once the number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribes. This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.
- This feature is also called as “Consumer Group”. In the same way, Kafka will provide the best of both the systems in a very simple and efficient manner.

Role of ZooKeeper

A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service. Zookeeper serves as the coordination interface between the Kafka brokers and consumers. The Kafka servers share information via a Zookeeper cluster. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster. Kafka will restore the state, once the Zookeeper restarts. This gives zero downtime for Kafka. The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.

3.5 Apache Cassandra

Apache Cassandra is highly scalable, high performance, distributed NoSQL database. Cassandra is designed to handle huge amount of data across many commodity servers, providing high availability without a single point of failure.

Cassandra has a distributed architecture which is capable to handle a huge amount of data. Data is placed on different machines with more than one replication factor to attain a high availability without a single point of failure.

NoSQLDatabase

A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

The primary objective of a NoSQL database is to have

- simplicity of design,
- horizontal scaling, and
- finer control over availability.

NoSql databases use different data structures compared to relational databases. It makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it must solve.

Cassandra Architecture

Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Data Replication in Cassandra

In Cassandra, nodes in a cluster act as replicas for a given piece of data. If some of the nodes are responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.

See the following image to understand the schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.

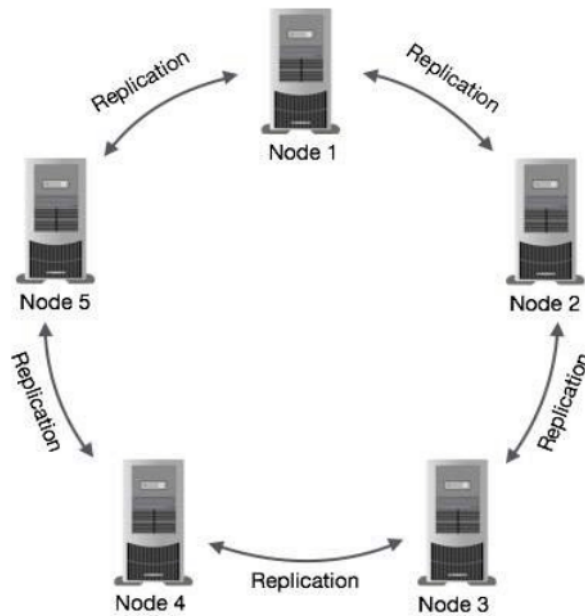


figure 10 : Node Replication

Components of Cassandra :

The main components of Cassandra are:

- Node: A Cassandra node is a place where data is stored.
- Data center: Data center is a collection of related nodes.
- Cluster: A cluster is a component which contains one or more data centers.
- Commit log: In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.

- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Query Language

Cassandra Query Language (CQL) is used to access Cassandra through its nodes. CQL treats the database (Keyspace) as a container of tables. Programmers use cqlsh: a prompt to work with CQL or separate application language drivers.

The client can approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

Write Operations

Every write activity of nodes is captured by the commit logs written in the nodes. Later the data will be captured and stored in the mem-table. Whenever the mem-table is full, data will be written into the SSTable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

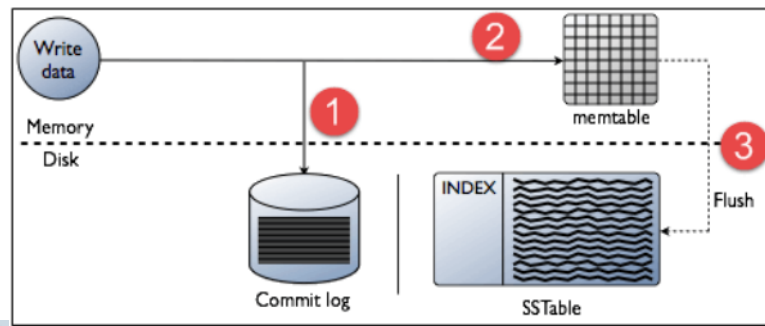


figure 11 : Cassandra Write Operation

Read Operations

In Read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable which contains the required data.

There are three types of read request that is sent to replicas by coordinators.

- Direct request
- Digest request
- Read repair request

The coordinator sends direct request to one of the replicas. After that, the coordinator sends the digest request to the number of replicas specified by the consistency level and checks if the returned data is an updated data.

After that, the coordinator sends digest request to all the remaining replicas. If any node gives out of date value, a background read repair request will update that data. This process is called read repair mechanism.

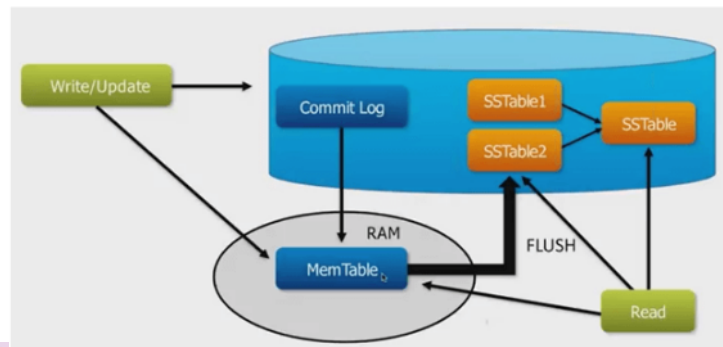


figure 12 : Read Operation

Cassandra Data Model

Data model in Cassandra is totally different from normally we see in RDBMS. Let's see how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that are operated together. The outermost container is known as the Cluster which contains different nodes. Every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. Following are the basic attributes of Keyspace in Cassandra:

- Replication factor: It specifies the number of machine in the cluster that will receive copies of the same data.
- Replica placement Strategy: It is a strategy which species how to place replicas in the ring. There are three types of strategies such as:
 - 1) Simple strategy (rack-aware strategy)
 - 2) old network topology strategy (rack-aware strategy)

3) network topology strategy (datacenter-shared strategy)

- Column families: column families are placed under keyspace. A keyspace is a container for a list of one or more column families while a column family is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

In Cassandra, a well data model is very important because a bad data model can degrade performance, especially when you try to implement the RDBMS concepts on Cassandra.

Cassandra data Models Rules

Cassandra doesn't support JOINS, GROUP BY, OR clause, aggregation etc. So you have to store data in a way that it should be retrieved whenever you want.

Cassandra is optimized for high write performances so you should maximize your writes for better read performance and data availability. There is a tradeoff between data write and data read. So, optimize you data read performance by maximizing the number of data writes. Maximize data duplication because Cassandra is a distributed database and data duplication provides instant availability without a single point of failure.

3.6 MODEL DEVELOPMENT

Flow-chart

Below is the flow chart that will describe the overall flow that Spring follows .

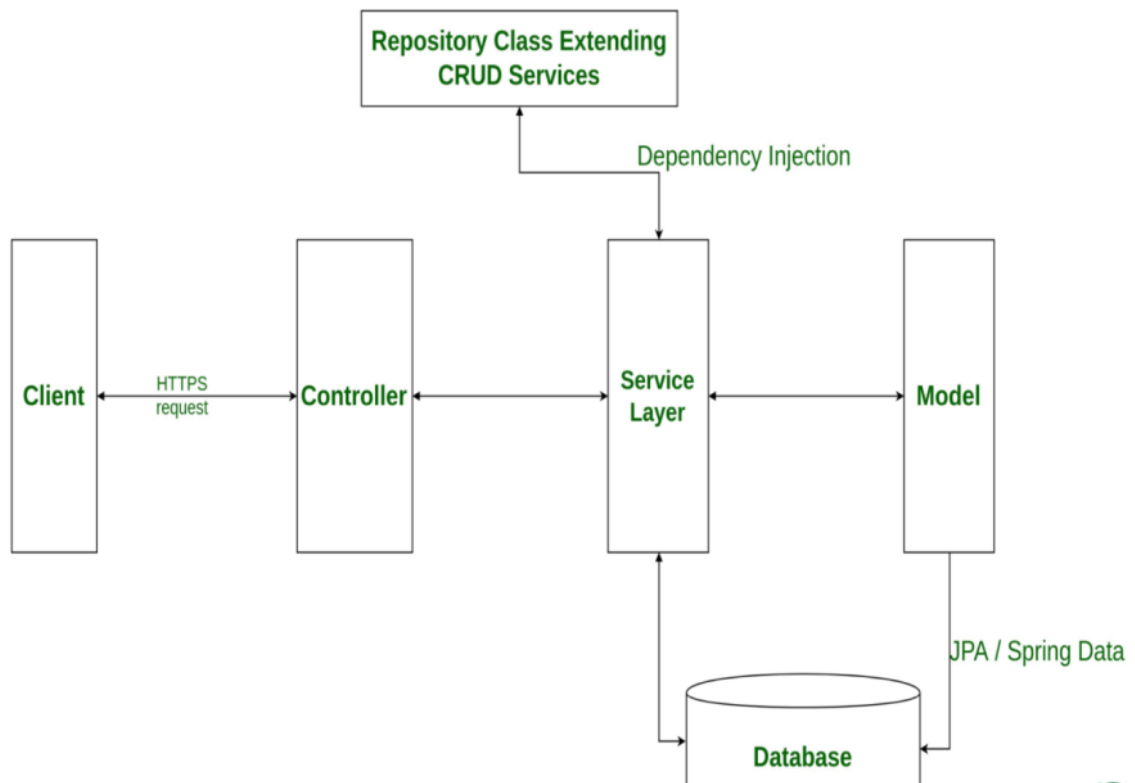
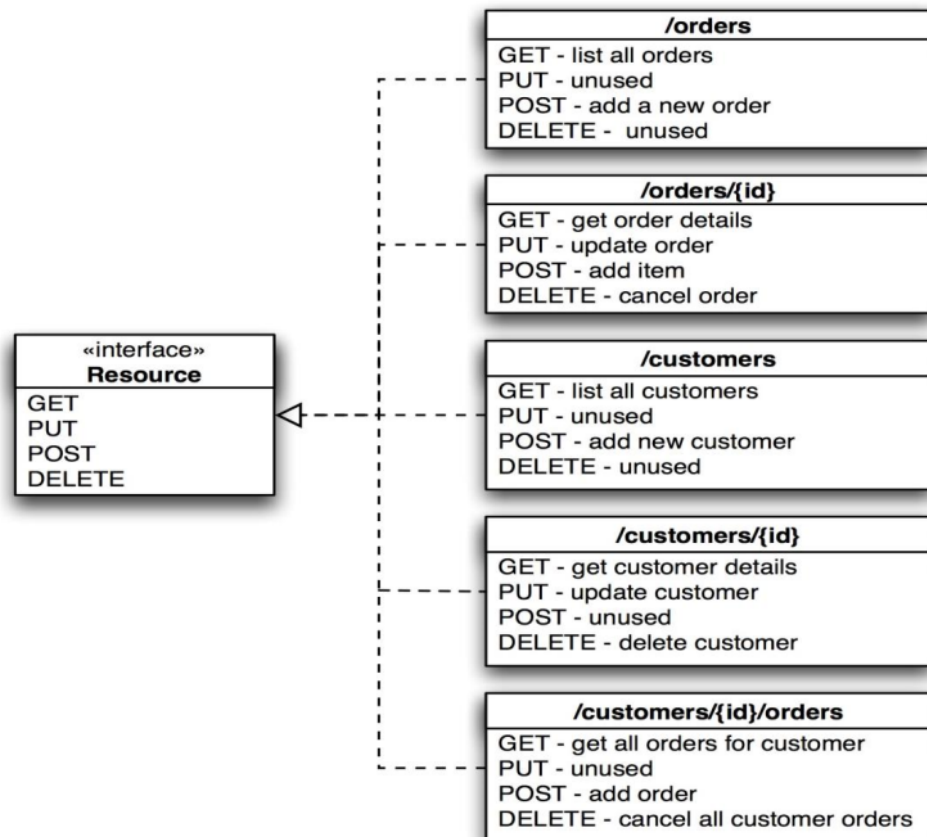


Figure 13 : flow chart

Schema diagram

Below is the Schema diagram of REST that will show what are the services available and who can access them.



1

Figure 14 : Schema diagram

3.7 Features

Reactive Programming solves performance problems caused by the use of native threads and the “One thread per request” paradigm. However, this solution goes along with higher development and maintenance complexity because testing and debugging, among other things, become more complicated.

Green threads are a possible way to avoid the performance losses caused by the process switches in the operating system. These were available in Java 1.1 but were already discarded in Java 1.3 because they did not allow the benefits of multi-core or multi-processor systems to be used. A new attempt to introduce another variant of Green Threads, so-called Fibers, into the JDK is Project Loom. This proposal would be accompanied by support for continuations in Java as a kind of spin-off product. This feature is known from other programming languages like Kotlin and Go under the name Coroutines

- avoid “callback hell”
- a lot simpler to do async / threaded work
- a lot of operators that simplify work
- very simple to compose streams of data
- complex threading becomes very easy
- you end up with a more cleaner, readable code base
- easy to implement back-pressure

■

■

CHAPTER 4

PERFORMANCE ANALYSIS

4.1 Starting Spring Boot Application

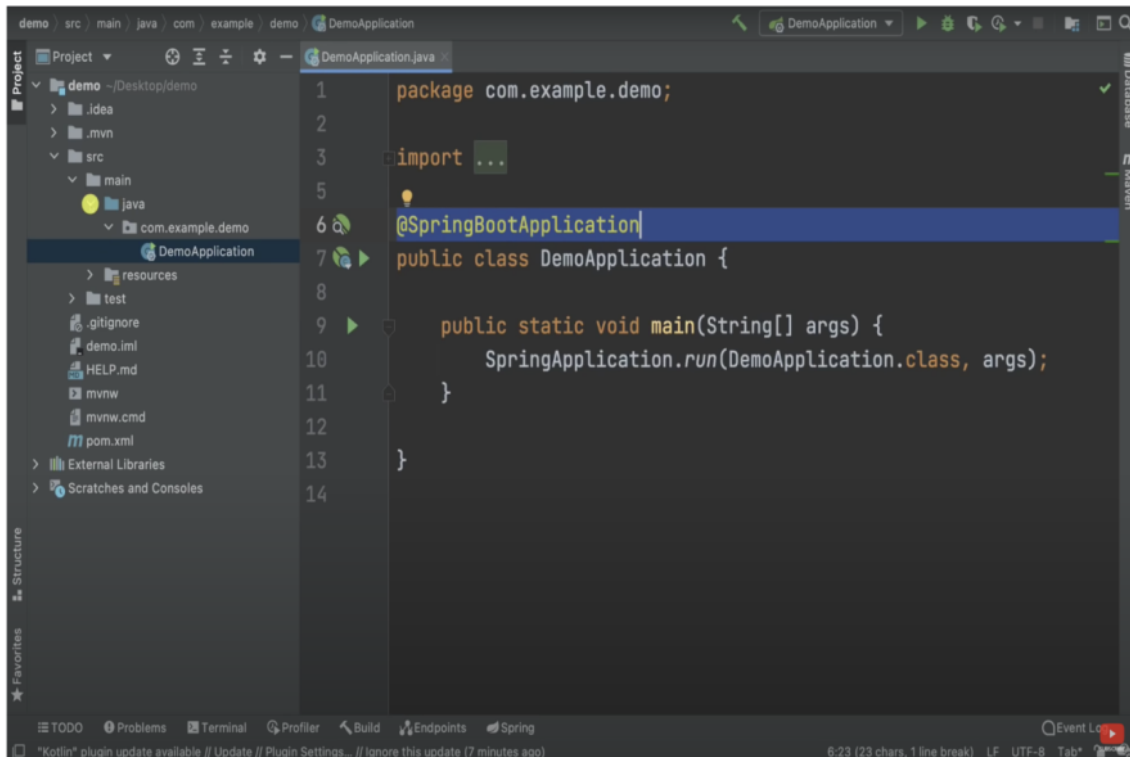
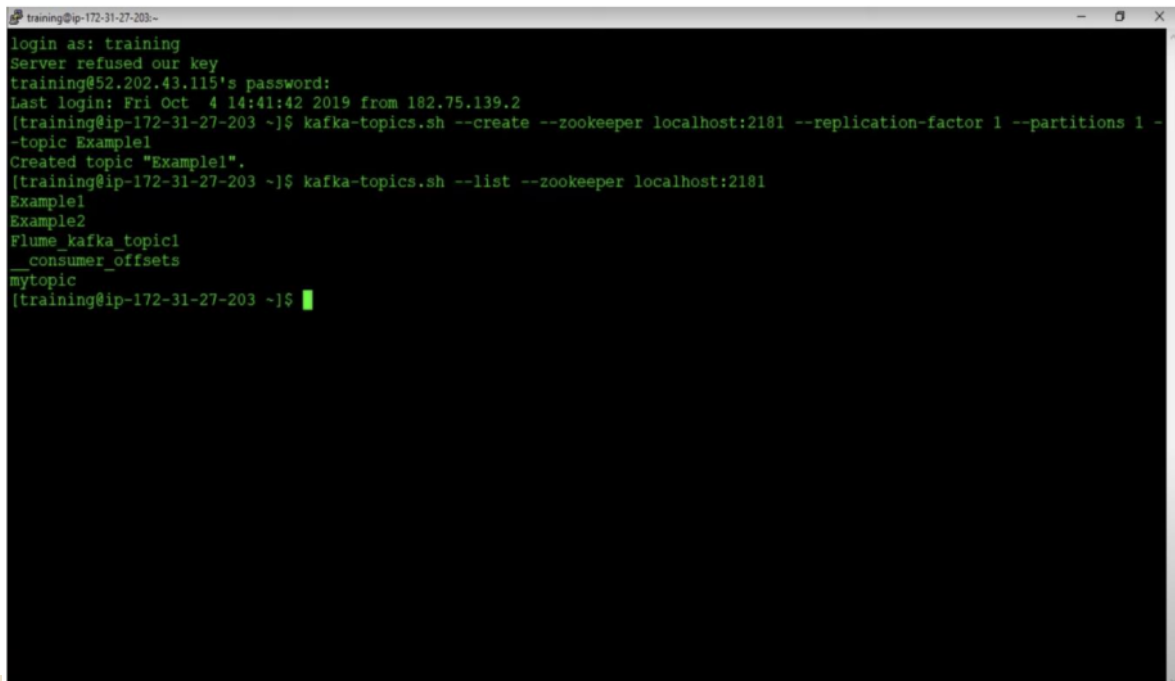


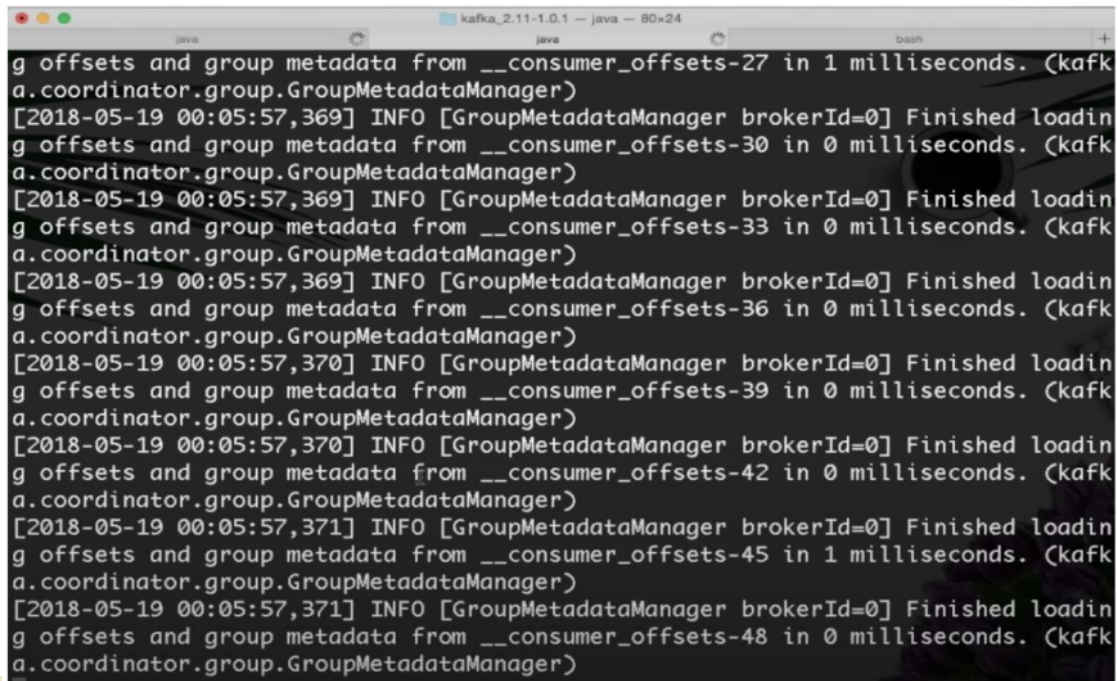
Figure 15: Spring Boot Application

4.2 Kafka



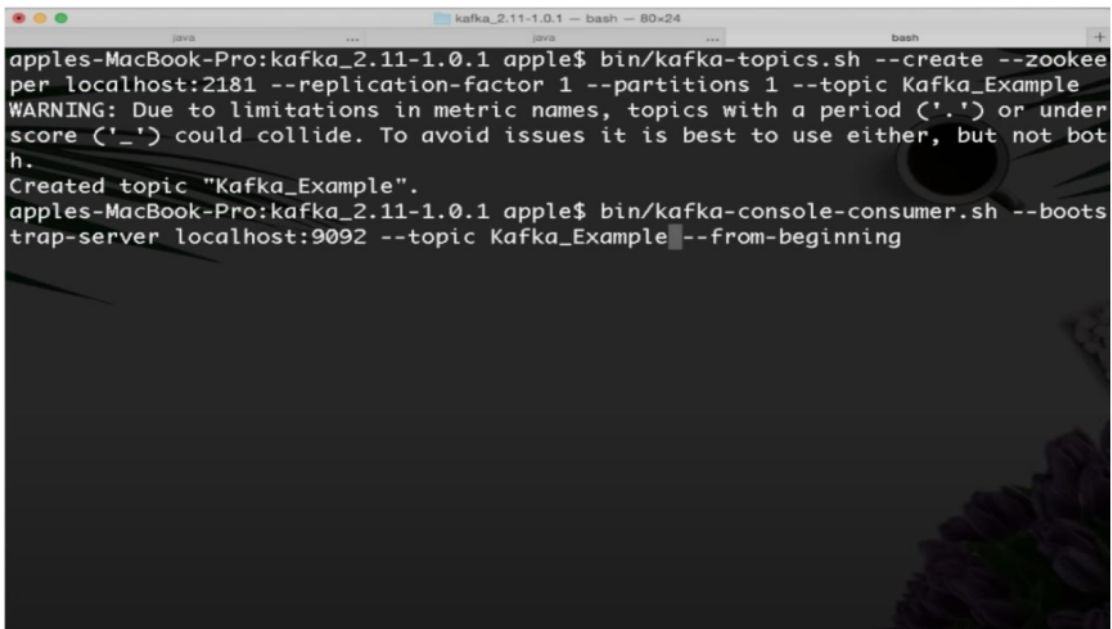
```
training@ip-172-31-27-203~  
login as: training  
Server refused our key  
training@52.202.43.115's password:  
Last login: Fri Oct  4 14:41:42 2019 from 182.75.139.2  
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Example1  
Created topic "Example1".  
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181  
Example1  
Example2  
Flume_kafka_topic1  
consumer_offsets  
mytopic  
[training@ip-172-31-27-203 ~]$
```

Figure 16:- Kafka Zookeeper

A screenshot of a terminal window titled 'kafka_2.11-1.0.1 -- java -- 80x24'. The terminal shows a series of log messages from the 'kafka.coordinator.group.GroupMetadataManager' broker. The messages indicate that the broker has finished loading group offsets and metadata for various consumer groups, including __consumer_offsets-27, -30, -33, -36, -39, -42, -45, and -48. Each message includes a timestamp of 2018-05-19 00:05:57 and a duration of 0 or 1 milliseconds.

```
g offsets and group metadata from __consumer_offsets-27 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-30 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-33 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,369] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-36 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,370] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-39 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,370] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-42 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,371] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-45 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-19 00:05:57,371] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-48 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```

Figure 17: Kafka Server

A screenshot of a terminal window titled 'kafka_2.11-1.0.1 -- bash -- 80x24'. The terminal shows the execution of two Kafka commands. The first command is 'bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Kafka_Example', which results in a warning about metric names and the creation of the topic 'Kafka_Example'. The second command is 'bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Kafka_Example --from-beginning', which is used to start consuming messages from the newly created topic.

```
apples-MacBook-Pro:kafka_2.11-1.0.1 apple$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Kafka_Example
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic "Kafka_Example".
apples-MacBook-Pro:kafka_2.11-1.0.1 apple$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Kafka_Example --from-beginning
```

Figure 18: Kafka Consumer

4.3 REST Controller

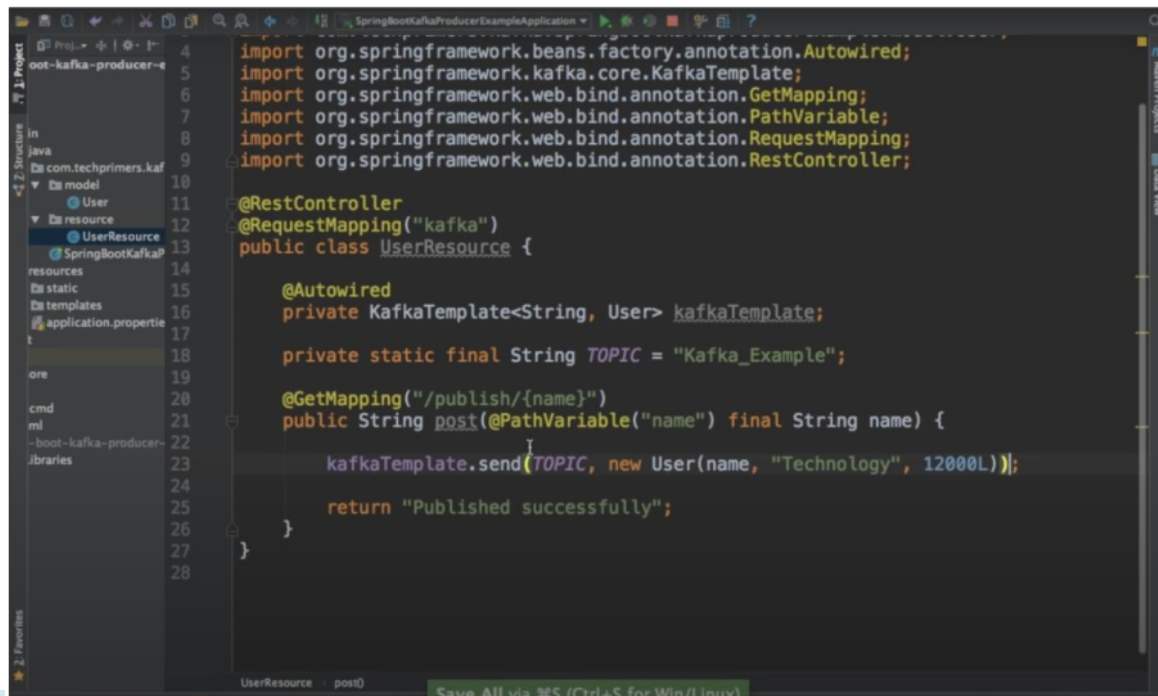


Figure 19: Apache sever Image.

4.1 Docker

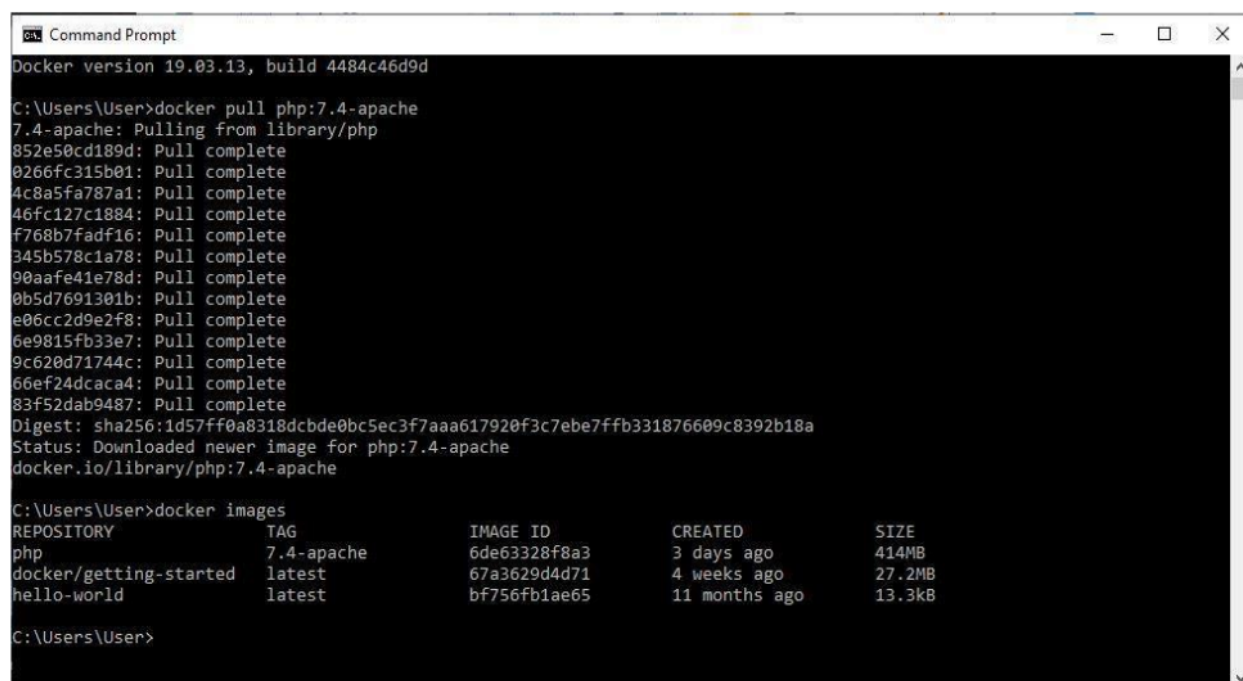


Figure 20: Docker Image

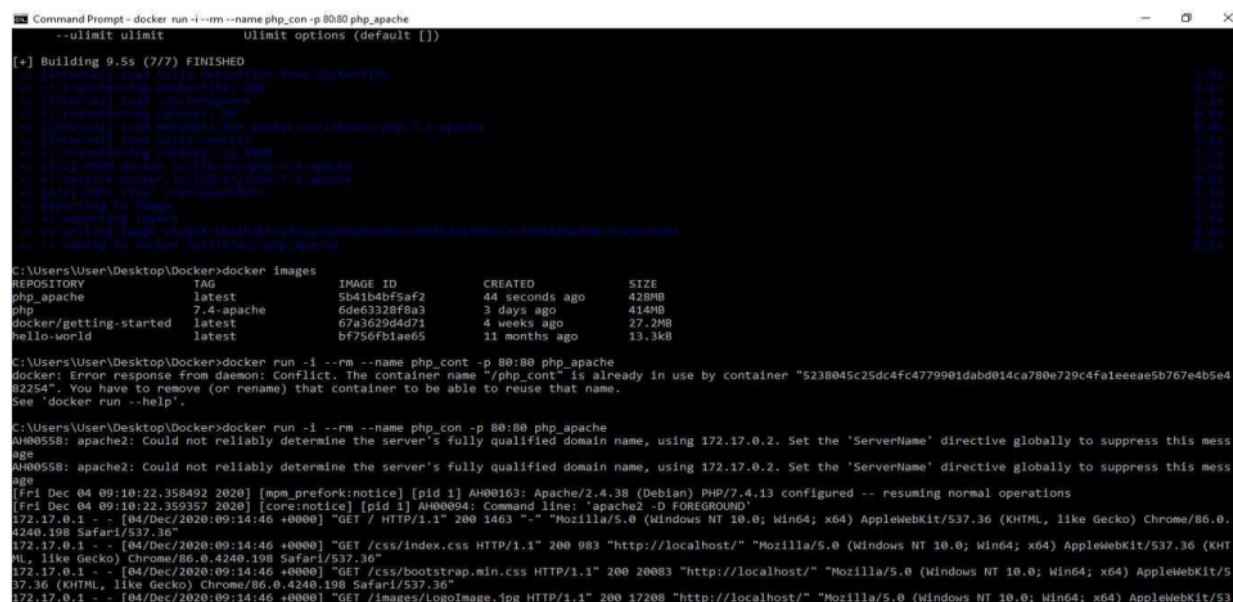


Figure 21: Building Container


```

C:\Users\User\Desktop\docker>docker-compose up
Creating network "docker_default" with the default driver
Building web
Step 1/3 : FROM php:7.4-apache
----> 6de63328f8a3
Step 2/3 : COPY site/ /var/www/html/
----> 4b4a1c3100cd
Step 3/3 : EXPOSE 80
----> Running in 593ea199d8ec
Removing intermediate container 593ea199d8ec
----> 3dab95ce19d9

Successfully built 3dab95ce19d9
Successfully tagged docker_web:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Pulling db (mysql:latest)...
latest: Pulling from library/mysql
852e58cd199d: Already exists
30969d0b0ffb: Pull complete
a43f41a44c48: Pull complete
5cdd802543a3: Pull complete
b79b040de953: Pull complete
938c64119969: Pull complete
7689ec51a0d9: Pull complete
a880ba7c411f: Pull complete
984f656ec6ca: Pull complete
9f497bce458a: Pull complete
b9940f97694b: Pull complete
2f069358dc96: Pull complete
Digest: sha256:4bb2e81a40e9d0d59bd8e3dc2ba5e1f2197696f6de39a91e90798dd27299b093
Status: Downloaded newer image for mysql:latest
Creating docker_web_1 ...
Creating docker_web_1 ... error
WARNING: Host is already in use by another container
ERROR: for docker_web_1 Cannot start service web: driver failed programming external connectivity on endpoint docker_web_1 (5b8dec2c56e50b81039f293d5b66ede472c7d55c086)
Creating docker_db_1 ... done
ERROR: for web Cannot start service web: driver failed programming external connectivity on endpoint docker_web_1 (5b8dec2c56e50b81039f293d5b66ede472c7d55c086ff093462125c4d9de4e81): Bind for 0.0.0.0:80 failed: port is already allocated
ERROR: Encountered errors while bringing up the project.

C:\Users\User\Desktop\docker>

```

Figure 22: Database Image

```

C:\Users\User\Desktop\docker>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
8c5145957d74   mysql    "docker-entrypoint.s..." 2 minutes ago  Up 2 minutes  0.0.0.0:3306->3306/tcp, 33060/tcp  docker_db_1
b1c09c0396fc   php_apache "docker-php-entrypoi..." 6 minutes ago  Up 6 minutes  0.0.0.0:80->80/tcp                php_con

C:\Users\User\Desktop\docker>docker exec -it 8c5145957d74 bash
root@8c5145957d74:/# mysql -uroot -p12345
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
-> ^C
mysql> show databases;
+-----+
| Database |
+-----+
| foodorder |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql>

```

Figure 23: Database runtime

CONCLUSION

When it comes to huge volumes of data or multi-userness, we often need asynchronous processing to make our systems fast and responsive. In Java, a representative of old object-oriented programming, asynchronicity can become really troublesome and make the code hard to understand and maintain. So, reactive programming is especially beneficial for this 'purely' object-oriented environment as it simplifies dealing with asynchronous flows. With its latest releases (starting with Java 8), Java itself has made some attempts to introduce built-in reactivity, yet these attempts are not very popular with developers to date. But there're some live and regularly updated third-party implementations for reactive programming in Java that help to save the day and thus are particularly loved and cherished by Java developers.

FUTURE WORK

In future we'll try and create the appliance serverless. The serverless computing model permits you to make and run applications and services while not having to concern infrastructure or servers. It eliminates infrastructure management tasks like server provisioning, patching, software system maintenance, scaling, and capability provisioning. Building native serverless applications implies that developers will specialise in the core product and on innovating applications and solutions, instead of outlay tons of your time on putting in and maintaining infrastructure.

REFERENCE

- [1] A survey on Reactive Programming: Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx and Wolfgang De Meuter
https://www.researchgate.net/publication/233755674_A_Survey_on_Reactive_Programming
- [2] Shantashree Das, Debomalya Ghose, Influence Of Fast Online Apps On The Operations Of The Restaurant Business : INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 8, ISSUE 12, DECEMBER 2019 ISSN 2277-8616
- [3] P.Nagendra Babu , M.Chaitanya Kumari , S.Venkat Mohan [3] has worked on computationof cloud computing, International Journal of Engineering Trends and Technology (IJETT)
– Volume 21 Number 6 – March 2015 ISSN: 2231-5381
- [4] BabakBashari Rad, Harrison John Bhatti and Mohammad Ahmadi [4] worked on Docker which provide some facilities, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 07 | July 2020

ZopSmart_Technology.pdf

ORIGINALITY REPORT

10%

SIMILARITY INDEX

10%

INTERNET SOURCES

6%

PUBLICATIONS

6%

STUDENT PAPERS

PRIMARY SOURCES

1

www.utm.edu

Internet Source

4%

2

www.darwinrecruitment.com

Internet Source

2%

3

cendana.com.bn

Internet Source

1%

4

archive.org

Internet Source

<1%

5

tools.ietf.org

Internet Source

<1%

6

images.autodesk.com

Internet Source

<1%

7

Submitted to ESCP-EAP

Student Paper

<1%

8

americanradiohistory.com

Internet Source

<1%

9

www.tandfonline.com

Internet Source

<1%

10	ethesis.nitrkl.ac.in Internet Source	<1 %
11	Submitted to Jaypee University of Information Technology Student Paper	<1 %
12	www.tenox.net Internet Source	<1 %
13	monada.com.ua Internet Source	<1 %
14	manualzz.com Internet Source	<1 %
15	www.netapp.com Internet Source	<1 %
16	guides.dss.gov.au Internet Source	<1 %

Exclude quotes On
Exclude bibliography On

Exclude matches < 10 words

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 17/6/2021

Type of Document (Tick): ☐ PhD Thesis ☐ M.Tech Dissertation/ Report ☒ B.Tech Project Report ☐ Paper

Name: Aditya Kumar Singh Department: CSE Enrolment No 171289

Contact No. 7209280425 E-mail. 171289@juitsolan.in

Name of the Supervisor: Dr. Monika Bharti Jindal

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): REACTIVE SPRING BOOT

APPLICATION FOR PRODUCT VARIANT USING KAFKA AND CASSANDRA DATABASE

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

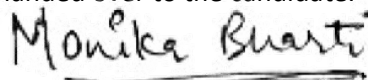
Complete Thesis/Report Pages Detail:

- Total No. of Pages = 55
- Total No. of Preliminary pages = 6
- Total No. of pages accommodate bibliography/references = 1

Aditya Kumar Singh
(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at10..... (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.



(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none">• All Preliminary Pages• Bibliography/Images/Quotes• 14 Words String	10%	Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com