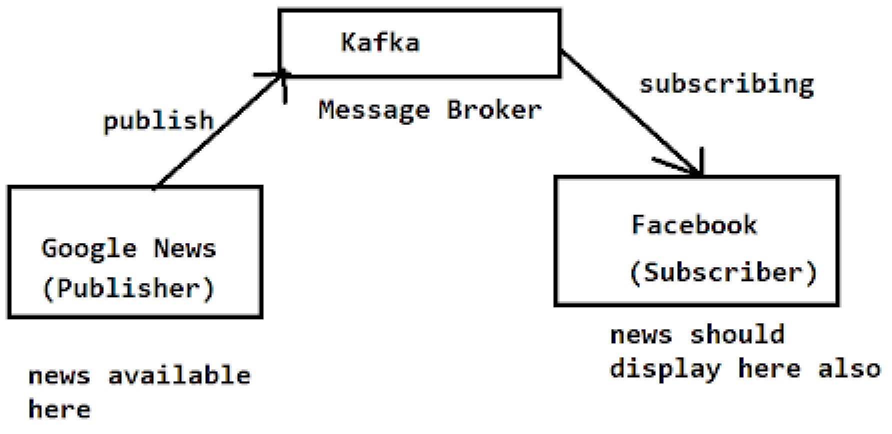
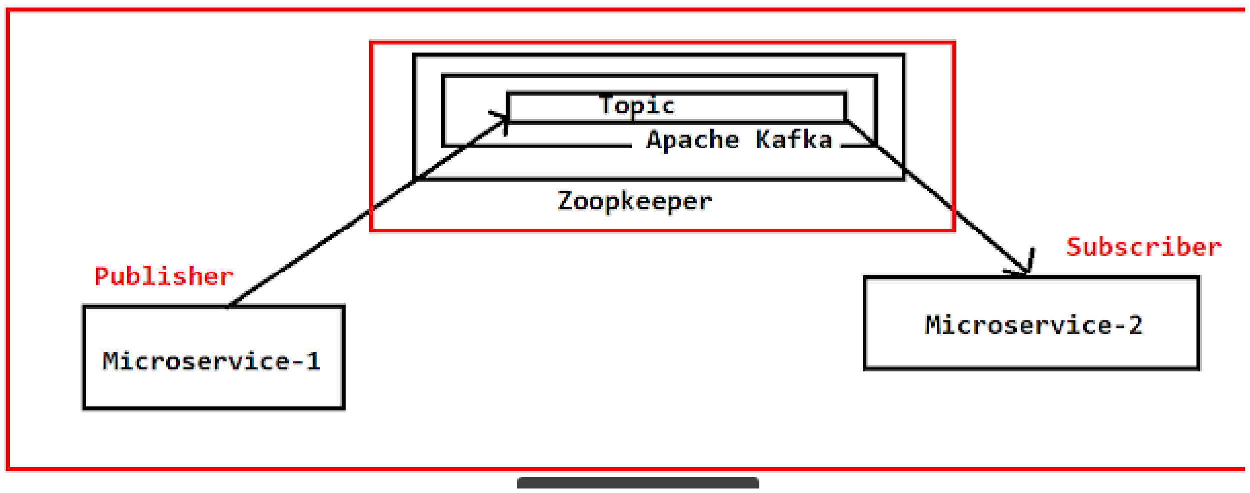
```
Apache Kafka
------------
Apache Kafka is an open-source stream-processing software platform
It is developed by the Apache Software Foundation

It written in Scala and Java.

The project aims to provide a unified, high-throughput, low-latency
platform for handling real-time data feeds.

Apache Kafka will act as Message Broker

When we are working with Apache Kafka, two parties will be available. one
application will acts as Publisher another application will act as
Subscriber.

Apache Kafka works based on publisher and subscriber model

If one application publish a message, mutliple applications can subscribe
to that message.
```

- ➔ When the data is coming continuously, then it is called data feeds.
  Ex: when we use news feedback on facebook , this is real time data
  Google news is generated by google which is coming on facebook continuously, this is called real time data feed

To use the kafka we need to install Zookeper.

How to display the google data in facebook?





What is event streaming?

Event streaming is the digital equivalent of the human body's central nervous system. It is the technological foundation for the 'always-on' world where businesses are increasingly software-defined and automated, and where the user of software is more software.

Technically speaking, event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events; storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed. Event streaming thus ensures a continuous flow and interpretation of data so that the right information is at the right place, at the right time.

What can I use event streaming for?

Event streaming is applied to a wide variety of use cases across a plethora of industries and organizations. Its many examples include:

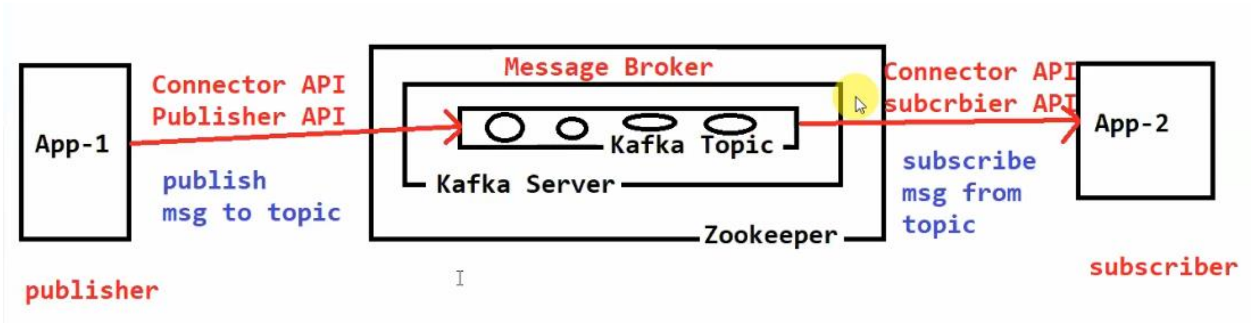
- To process payments and financial transactions in real-time, such as in stock exchanges, banks, and insurances.
- To track and monitor cars, trucks, fleets, and shipments in real-time, such as in logistics and the automotive industry.
- To continuously capture and analyze sensor data from IoT devices or other equipment, such as in factories and wind parks.
- To collect and immediately react to customer interactions and orders, such as in retail, the hotel and travel industry, and mobile applications.
- To monitor patients in hospital care and predict changes in condition to ensure timely treatment in emergencies.
- To connect, store, and make available data produced by different divisions of a company.
- To serve as the foundation for data platforms, event-driven architectures, and microservices.

Apache Kafka® is an event streaming platform. What does that mean?

Kafka combines three key capabilities so you can implement your use cases for event streaming end-to-end with a single battle-tested solution:

1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
2. To **store** streams of events durably and reliably for as long as you want.
3. To **process** streams of events as they occur or retrospectively.

For more learning visit: https://kafka.apache.org/intro



There are so many message broker are available in market. Like JMS, Active MQ, Rabit MQ, Kafka

```
APache Kafka is a distributed streaming platform

Apache Kafka is used to process real time data feeds with high throughput
and low latency

Ex : flights data, sensors data, stocks data, news data etc....

Kafka works based on Publisher and Subscriber model

Kafka Terminology
-----------------
```

Zookeeper
Kafka Server
Kafka Topic
Message
Publisher
Subscriber

Kafka APIs

Connector API
Publisher API
Subscriber API
Streams API
Spring Boot + Apache Kafka Application
=======================================

Step-1 : Download Zookeeper from below URL

   URL : http://mirrors.estointernet.in/apache/zookeeper/stable/

Step-2 : Download Apache Kafka from below URL

   URL : http://mirrors.estointernet.in/apache/kafka/

Step-3 : Set Path to ZOOKEEPER in Environment variables upto bin folder

Step-4 : Start Zookeeper server using below command from Kafka folder

```
    Command : zookeeper-server-start.bat zookeeper.properties
```

Note: Above command will available in kafka/bin/windows folder
Note: zookeeper.properties file will be available in config folder. You can copy zookeeper.properties and server.properties files from kafka/config folder to kafka/bin/windows folder.

Step-5: Start Kafka Server using below command from Kakfa folder

```
    Command : kafka-server-start.bat server.properties
```

Note: server.properties file will be available in config folder (Copied to windows folder)

Step-6 : Create Kakfa Topic using below command

```
Command : kafka-topics.bat --create --zookeeper localhost:2181 --
replication-factor 1 --partitions 1 --topic 03-sbms-topic
```

Step-7 : View created Topics using below command

```
      Command : kafka-topics.bat --list --zookeeper localhost:2181
```

Step-8 : Create Spring Boot Project in IDE
===

Step-9: Add below kafka related dependencies in pom.xml

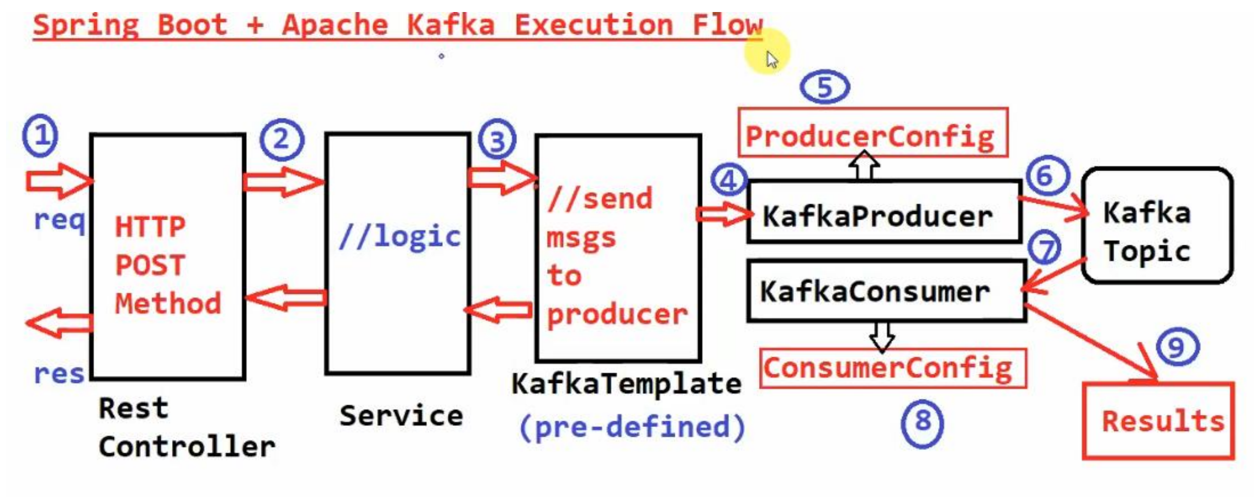
```
		<dependency>
			<groupId>org.apache.kafka</groupId>
			<artifactId>kafka-streams</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.kafka</groupId>
			<artifactId>spring-kafka</artifactId>
		</dependency>
		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-databind</artifactId>
		</dependency>
```

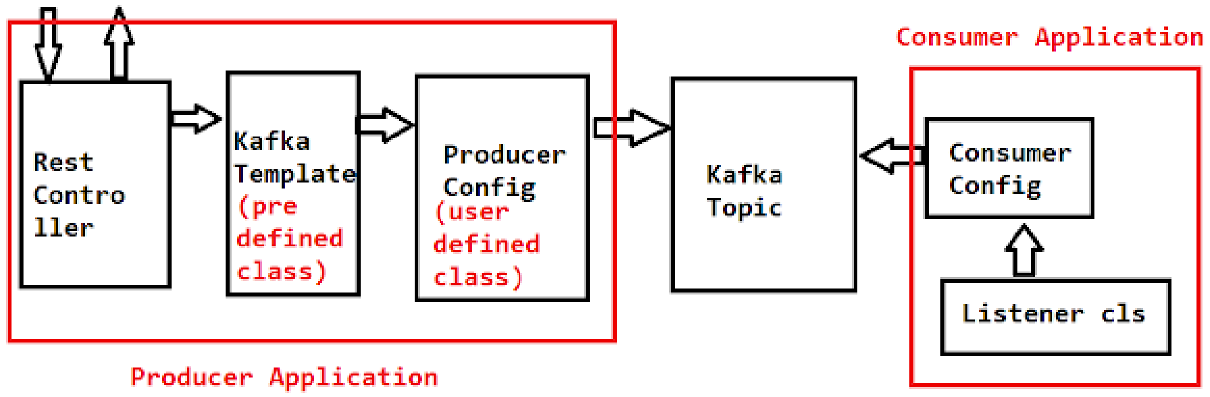
Step-9: Create RestController, KafaProducer and KafaConsumer classes to publish and subsribe message

Step-10: Test application using PostMan.

Sample Data

```
{
"customerId":101,
"customerName":"Ashok",
"customerEmail":"ashok@gmail.com"
}
---------------------------------
[

{"customerId":101,
"customerName":"Ashok",
"customerEmail":"ashok@gmail.com"
},

{
"customerId":102,
"customerName":"Raj",
"customerEmail":"raj@gmail.com"
},
{
"customerId":102,
"customerName":"John",
"customerEmail":"john@gmail.com"
}

]
```





```
Steps to develop Spring Boot application to work with Kafka
-------------------------------------------------------------
1) Start Zookeeper server

2) Start Apache Kafka Server

3) Create Spring Boot application with below dependencies

        1)spring-boot-starter-web
        2)project lombok
        3)swagger
        4)kafka-streams
        5)spring-kafka
        6)jackson-databind

<dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.apache.kafka</groupId>
                        <artifactId>kafka-streams</artifactId>
```

```
		</dependency>
		<dependency>
			<groupId>org.springframework.kafka</groupId>
			<artifactId>spring-kafka</artifactId>
		</dependency>

		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-databind</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.kafka</groupId>
			<artifactId>spring-kafka-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
```

4) Create ProducerConfig class and Configure KafkaTemplate class as Spring Bean by injection KafkaProducerFactory bean.

```
package com.ashok.config;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;

import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;

/**
 *
 * @author Ashok
 *
 */
@Configuration
public class KafkaProduceConfig {

	/**
	 * This method is used to Kafka Producer Config details
```

```
         * @return
         */

        @Bean
        public ProducerFactory<String, Customer> producerFactory() {

                Map<String, Object> configProps = new HashMap<String,
Object>();

                configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
KafkaConstants.HOST);
                configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);

        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
JsonSerializer.class);

                return new DefaultKafkaProducerFactory(configProps);
        }

        /**
         * This method is used to create KafkaTemplate bean obj
         * @return
         */
        @Bean(name = "kafkaTemplate")
        public KafkaTemplate<String, Customer> kafkaTemplate() {
                return new KafkaTemplate<>(producerFactory());
        }

}
```

5) Create Service class and inject KafkaTemplate into it. KafkaTemplate class provided method to publish a message to topic.

```
@Service("customerService")
public class CustomerService {

        @Autowired
        private KafkaTemplate<String, Customer> kafkaTemplate;

        /**
         * This method is used to publish customer records as msgs to kafka
topic
         *
         * @param customers
         * @return
         */
        public String add(List<Customer> customers) {

                if (!customers.isEmpty()) {
                        for (Customer c : customers) {
                                kafkaTemplate.send(KafkaConstants.TOPIC, c);
```

```
                              System.out.println("************Msg published
to Kafka topic**************");
                      }
              }
              return "Customer Record Added To Kafka Queue Successfully";
       }

}
```

6) Create RestController and expose Distributed methods to publish message to kafka. RestController method will take customers data and will give it to Service class method. Service class method will call KafkaTemplate class send method then it will publish messag to kafka topic.

```
@RestController
public class CustomerRestController {

       @Autowired
       private CustomerService customerService;

       /**
        * This method is used to Customer records in post request
        * @param customers
        * @return
        */
       @PostMapping(value = "/addCustomer",
                      consumes = {
                                     MediaType.APPLICATION_JSON,
                                     MediaType.APPLICATION_XML
                      }
       )
       public String addCustomer(@RequestBody List<Customer> customers) {
              return customerService.add(customers);
       }

}
```

7) Configure Port Number in application.yml file

8) Run Spring Boot Application and Test it.

Developing Kafka Listener Project

-> Create KafkaListener config class

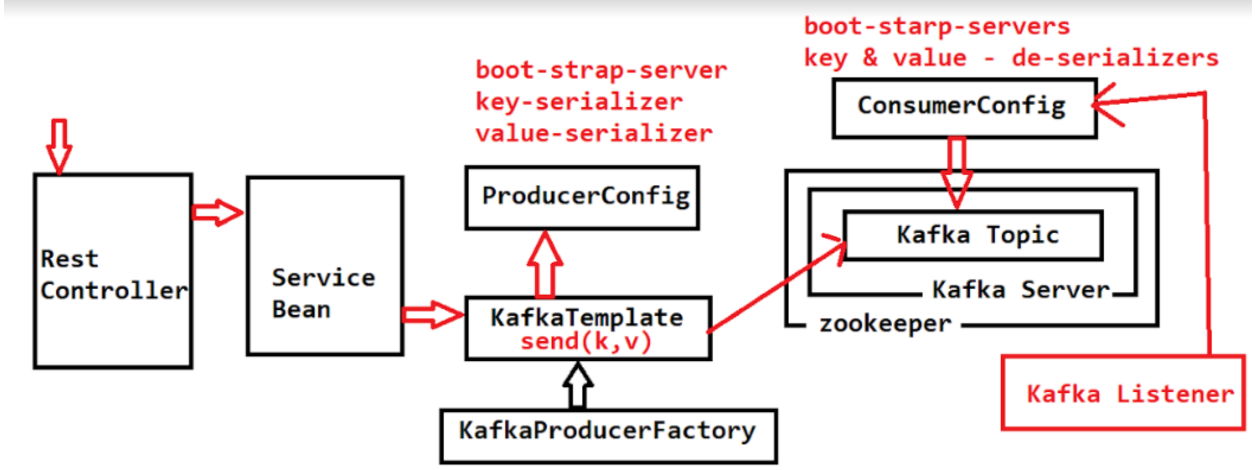
```
package com.ashok.config;
```

```
import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import
org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;

import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;

/**
 *
 * @author Ashok
 *
 */

@Configuration
@EnableKafka
public class KafkaListenerConfig {

        /**
         * This method is used to Kafka Consumer Config details
         *
         * @return
         */
        @Bean
        public ConsumerFactory<String, Customer> consumerFactory() {
                Map<String, Object> props = new HashMap();
                props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
KafkaConstants.HOST);
                props.put(ConsumerConfig.GROUP_ID_CONFIG,
KafkaConstants.GROUP_ID);
                props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
                props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
JsonDeserializer.class);

                return new DefaultKafkaConsumerFactory<>(props, new
StringDeserializer(),
                                new JsonDeserializer<>(Customer.class));
        }

        @Bean
        public ConcurrentKafkaListenerContainerFactory<String, Customer>
kafkaListenerContainerFactory() {
                ConcurrentKafkaListenerContainerFactory<String, Customer>
factory = new ConcurrentKafkaListenerContainerFactory<String, Customer>();
```

```
                factory.setConsumerFactory(consumerFactory());
                return factory;
        }

}
```

2) Write the method to recieve message from kafka topic

```
/**
         * This method is used to consume messages from kafka topic
         *
         * @param c
         * @return
         */
        @KafkaListener(topics = KafkaConstants.TOPIC, groupId =
KafkaConstants.GROUP_ID)
        public Customer listener(Customer c) {
                System.out.println("***Msg recieved from Kafka Topic ::" +
c);
                return c;
        }
```


boot-starp-servers
key & value - de-serializers
boot-strap-server
key-serializer
value-serializer
ConsumerConfig
ProducerConfig
Rest Controller
Service Bean
Kafka Topic
Kafka Server
zookeeper
KafkaTemplate
send(k,v)
KafkaProducerFactory
Kafka Listener