Microservices(Architecture)
==========
1. Difference b/w Monolith and Microservice Architecture
2. LoadBalancing(ClientSide(Ribbonclient) and ServerSide[LBR])
3. Eureka Server and DiscoverEureka Client[netflix provider]
4. InterService-Communication(FeignClient) vs Intra-Service Communication[RestTemplate,WebClient]
5. Actuators
6. SpringBoot-Admin Server and SpringBoot Admin Client
7. SpringCloud(Connnecting the microservices to Github)
8. Api-Gateway(Zuul proxy)
9. Distrubted logging[Zipkin Server and Sleuth]
10. Caching[RedisCache]
11. CircuitBreaker[Hystric Circuit Breaker]
12. EmbededDatabase[h2]
18. Swagger(Response Testing]
++
13. Apache-Kafka Integration
14. Spring-Reactive Programming[Mono and Flux Object]

MessageBroker
==========
  It is a software which is used to exchange the messages.
  It is used to give the data to another microservice.
  In real time scenarios the data keeps coming from services, to exchange the services we use "Message Broker".
   a. Apachekafka(latest used in Industry)
   b. RabbitMQ
   c. JMS
   d. ActiveMQ

=> To run ApacheKafka,we need to start ZooKeeper software and we need to create a topic.

=> Apache Kafka is a Streaming API which is used to process real data feeds with high throughput and low latency.
                eg: crickbuz score, train movement,flight movement information,stock market information.

=> It is developed by Linkedin and donated to Apache Organisation
=> It is implemented in Scala and Java Programming language.
=> It is an Open source
=> Apache Kafka architecture is based on Publisher/Subscriber model.

Terminologies associated with ApacheKafka
==========
 1. ZooKeeper(It is a software which provides support system/environment to run kafka server)
 2. Apache kafka Server(It is a message broker where the message(data) will be stored)
 3. Apache kafka Topic(it is a queue in kafka server where the message(data) will be stored)
 4. Apache Kafka Producer(Publisher)(It is the one which keeps the data in the queue)
 5. Apache Kafka Consumer(Consumer)(It is the one which consumes the data from the queue)

Apache Kafka internally uses 4 apis
==========
  a. Producer API => It is used to publish messages to kafka topic.
  b. Consumer API => It is used to read messages from Kafka topic.
  c. Connector API => It is used to connect both producer and consumer to kafka topic.
  d. Streams API   => It is used to read msgs from topic and convert them to output results(java obj -> json and json ->java obj).

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(zoopkeep server is running on portno::2181)

    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 kafka/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 kafka/config folder (Copied to kafka/bin/windows folder)

Step-6 : Create Kakfa Topic using below command from kafka/bin/windows folder

Command : kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic octbatch-ineuron-topic

Step-7 : View created Topics using below command

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

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 subscribe message
Step10: Run the application and test it through POSTMAN

Sample Data

```
[
 {
	"customerId":101,
	"customerName":"navin",
	"customerEmail":"navin@gmail.com"
 }
]
```

```
[

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

	{
		"customerId":102,
		"customerName":"hyder",
		"customerEmail":"hyder@gmail.com"
	},
	{
		"customerId":103,
		"customerName":"nitin",
		"customerEmail":"nitin@gmail.com"
	}
]
```

refer:: SB-apache-kafka-app

Mono and flux Object
====================

1. They are used in reactive programming
2. As a part of RestApi, we are using Mono and Flux Objects
3. Mono => Single response and Flux => Stream of responses.

Steps to create Rest API using Mono and Flux Objects
==

1. To work with Mono and Flux we will use web flux starters.

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Note: Web Flux supports reactive programming which was introduced in Spring 5.x version

2. Creating one binding class for Response

```
	@Data
	public class CustomerEvent {

		private String customerName;
		private Date date;

	}
```

3. Creating a RestController with 2 EndPoint methods
	a. Mono Response
	b. Stream Response

```
@RestController
public class CustomerRestController {

	@GetMapping(value = "/getEvent", produces = "application/json")
	public ResponseEntity<Mono<CustomerEvent>> getCustomerEvent() {

		// Creating Pojo object with data
		CustomerEvent event = new CustomerEvent("John", new Date());

		//Mono object which is used to send the response
		Mono<CustomerEvent> customerMono = Mono.just(event);

		ResponseEntity<Mono<CustomerEvent>> responseEntity =
				new ResponseEntity<Mono<CustomerEvent>>(customerMono, HttpStatus.OK);

		return responseEntity;
	}


	@GetMapping(value = "/getEvents", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
	public ResponseEntity<Flux<CustomerEvent>> getCustomerEvents() {

		// Creating Pojo object with data
		CustomerEvent event = new CustomerEvent("Smith", new Date());

		// Creating Stream object to send data
		Stream<CustomerEvent> customerStream = Stream.generate(() -> event);

		// Giving Stream object to Flux object
		Flux<CustomerEvent> ceFlux = Flux.fromStream(customerStream);

		// Setting Response Interval
		Flux<Long> interval = Flux.interval(Duration.ofMillis(500));

		// Combining IntervalFlux and CustomerEventFlux
		Flux<Tuple2<Long, CustomerEvent>> zip = Flux.zip(interval, ceFlux);

		// Getting Second Tuple value as Flux Obj
		Flux<CustomerEvent> fluxMap = zip.map(Tuple2::getT2);
```

```
            // Adding Flux Object to Response Body
            ResponseEntity<Flux<CustomerEvent>> resEntity =
                        new ResponseEntity<Flux<CustomerEvent>>(fluxMap,
HttpStatus.OK);

            // Returning ResEntity with Flux
            return resEntity;
      }

}
```

4. Configure port no in application.properties file and run boot application

```
      server.port = 9999
```

5. Test RestApi methods for the functionality.

```
       mono response url : http://localhost:9999/getEvent
      flux response url : http://localhost:9999/getEvents
```

refer:: SB-mono-flux-App