**Apache Kafka**

**Note**: Please refer the below link for producing the messages into kafka topic.

https://docs.spring.io/spring-kafka/reference/html/#sending-messages

**Setting Up Zookeeper and Kafka Broker**

**Setting up Kafka in local:**

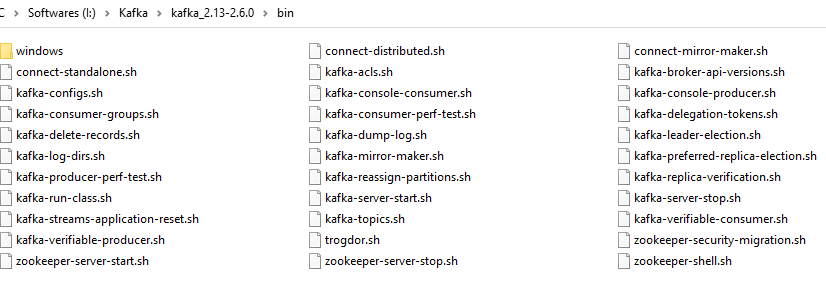
**Note:** We are going to spin-up zookeeper instance; zookeeper comes as part of Kafka binary.

**Step1**: launch the zookeeper instance

I:\Kafka\kafka\_2.13-2.6.0>dir

**LICENSES** **NOTICE** **BIN** **CONFIG** **LIBS** **LOGS** **SITE**-**DOCS**

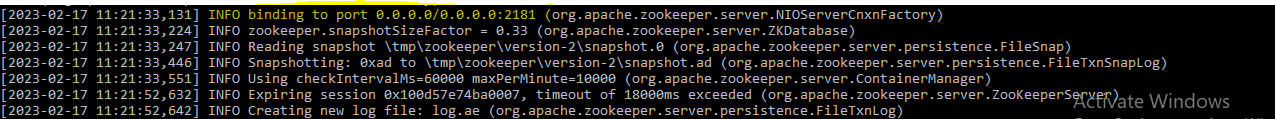
I:\Kafka\kafka\_2.13-2.6.0\bin>



**Note**: When we work on windows, we have to work on bat file for this we have to navigate to windows folder.

**I**:\kafka\kafka\_2.13-2.6.0\bin\windows>zookeeper-server-start.bat ..\..\config>zookeeper.properties





**Note:** with this we have zookeeper instance is ready. Zookeeper is running on **2181** port **.**

After zookeeper launch, launch the Kafka broker and Kafka broker register with zookeeper.

Zookeeper plays a vital role when we do have multiple Kafka brokers.

**Note**: Zookeeper manages the Kafka broker.

**Note**: Before start the Kafka brokers we need to add below configurations in **server.properties** file**.**

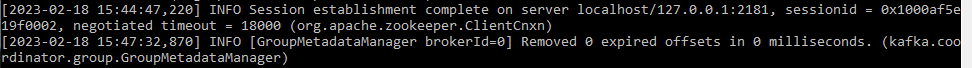
**listeners = PLAINTEXT://localhost:9092**

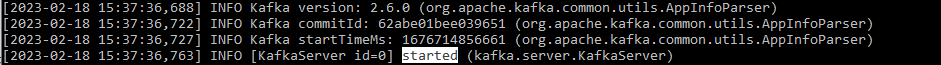
**auto.create.topics.enable=false**

**Step2: Start up the Kafka Broker**

I:\Kafka\kafka\_2.13-2.6.0\bin\windows>kafka-server-start.bat I:\Kafka\kafka\_2.13-2.6.0\config\server.properties

****

****

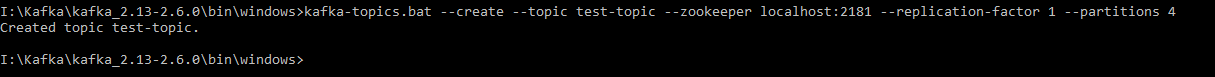
****

**Note:** Once Kafka broker is started it automatically registers with zookeeper. **[port 2181]**

**Step3: Create Kafka Topic and Produce and Consume Message From CLI.**

**3.1: Create Topic**

I:\Kafka\kafka\_2.13-2.6.0\bin\windows>kafks-topics.sh --create --topic test-topic --zookeeper localhost:2181 --replication-factor 1 –partitions 4

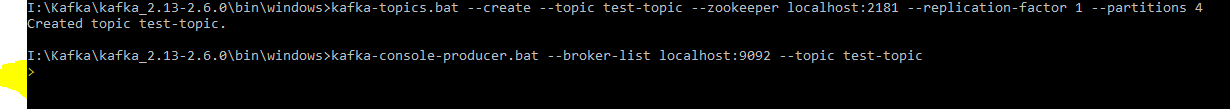
**Output:** Created Topic test-topic

**3.2: Start Kafka Producer Console in new window**

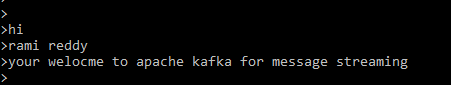
**Note: Without Key**

When we publish the messages to Kafka the messages without key then message will store in partition in round robin algorithm.

bin\windows>kafka-console-procedure.bat --broker-list locahost:9092 --topic test-topic



**Note**: When you see the above highlighted arrow symbol that means successfully producer instantiated and now we can produces messages to topic **test-topic.**

****

I have pushed above three messages into topic **test-topic:** now it’s time to consume messages from topics for this follow 3.3.

**3.3 Start Kafka Consumer Console in new window.**

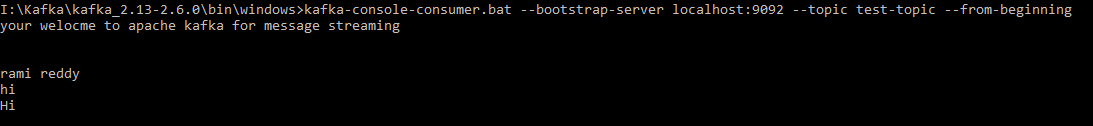
**Note: Without Key:**

When we produce the messages to Kafka without key, then consumer will consume the messages in descending order

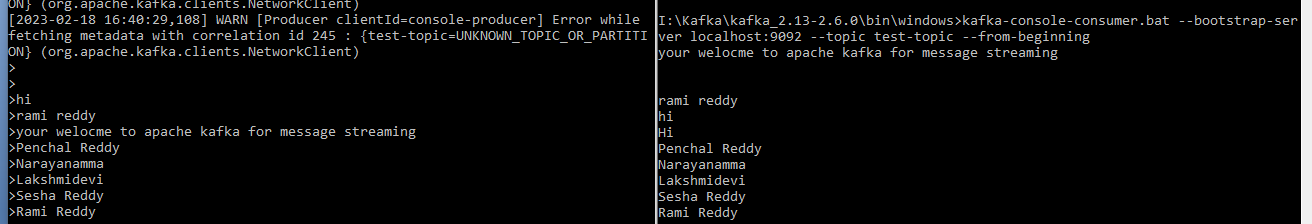
**bin\winndows\>**kafka-console.bat –bootstrap-server localhost:9092 –topic test-topic --from-beginning

**Note:** --from-beginning command will read the messages from beginning of the topic.

**Note:** The movement when you hit the enter from the command below, you start consuming the messages from topic **test-topic**

****

**Note: From Producer Side From Consumer Side**

****

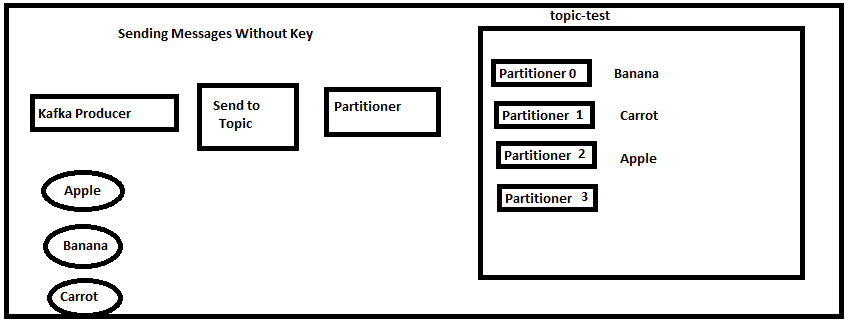
**Kafka Message:**

Kafka Message these sent from produces has **TWO** properties.

Key (Optional)

Value

**Note**: When we produce the messages to Kafka without key then partition will store the messages in round robin algorithm and consumer will consume the messages in descending order. Like below diagram.



**With Key: Messages are read in sequential manner.**

**Note: Produce the messages into Kafka topic using KEY**

bin\windows>kafka-console-procedure.bat --broker-list locahost:9092 --topic test-topic --property "key.separator=-" --property “parse.key=true”

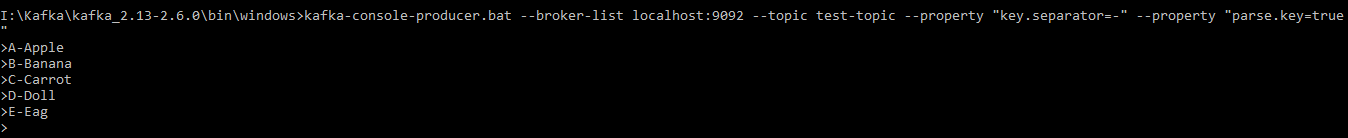
Example:

A-Apple

B-Bananna

C-Carrot

D-Doll

E-Eag****

**Note: Consume the messages from Kafka topic using KEY**

bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test-topic --from-beginning --property "key.separator= - " --property "print.key=true"

Examples:

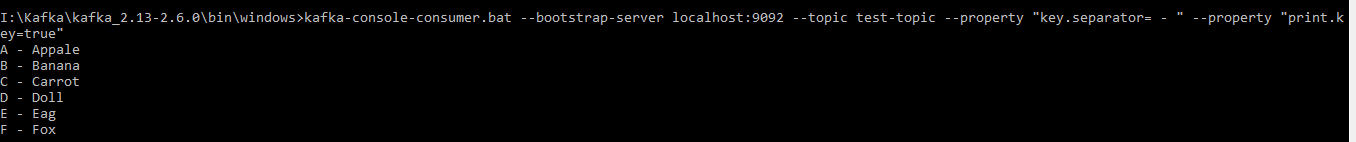
A - Apple

B - Bananna

C - Carrot

D - Doll

E - Eag

****

**Consumer OFFSETS:**

Consumers have three options to read message from Kafka topic.

1. --from-beginning
2. Latest
3. Specific offset

For any consumer it’s required the group id to read the messages from the topic.

Consumer can poll or read the messages from topic at the same time.

As a process each message it moves the consumer offset one by one

Let’s say some reason consumer is down or crashed. While consumer is down, producer produces the message into topic and now consumer is brought-up after sometime How does it consumer knows that reads from offset 6. Consumer Internally stores an internal topic called \_\_consumer\_\_offset.

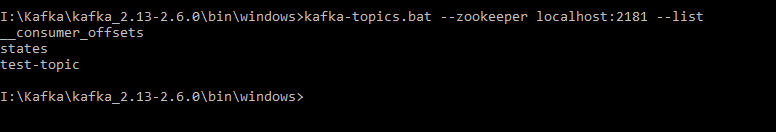
Repeating : Consumer once polls the all records from the topic commits the offset into the topic called \_\_consumer\_\_offset with the group id. Now even consume goes down after some time due to some reason and after it brought-up and consumer knows that from where to start read the message using topic called \_\_consumer\_\_offsets with group id.

**Note:** Consumer are behaves like bookmark for the consumer to start reading the messages from the point it left off.

**Where to check the \_\_consumer\_\_offset create**

**Step1:** first list the topics created

>kafka-topics.bat –zookeeper –localhost:2181 --list

****

**Note:** Here we have total three topics out of three **\_\_consumer\_offsets** created by the broker.

And which is the takes care of maintaining the consumer offset for us.

**Consumer Groups:**

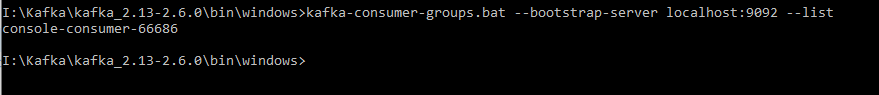
* Group.id is mandatory
* Group.id plays a major role when it comes to scalable message consumption
* Each different application will have a unique group id.
* Who manages the Consumer-group.

Kafka broker manages the consumer-groups

Kafka broker act as a Group Co-Ordinator.

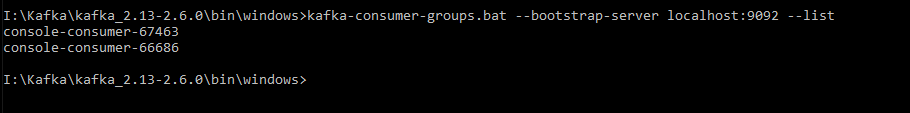
**How to View Consumer Groups:**

Kafka\bin\windows>kafka-consumer-group.bat –bootstrap-server localhost:9092 –list



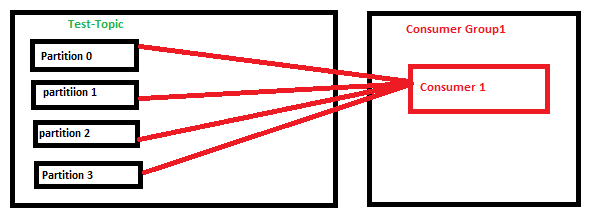
Note: When you run the console-consumer group it automatically creates the consumer-group for us.

* For each new console-consumer Kafka creates a unique group id for us.
* If we want to see the consumer group id’s run the following command twice in different command prompt.(this means two consumer instance we are going to spinning up for faster consumption purpose)
* **Bin\windows>kafka-console-consumer.bat –bootstrap-server localhost:9092**
* **Bin\windows>kafka-console-consumer.bat –bootstrap-server localhost:9092**
* Now run the kafka-consumer-group.bat to see list of consumer groups that created.

****

**Kafka Consumer Concepts**

* https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html
* In order to understand how to read data from Kafka, you first need to understand its consumers and consumer groups. The following sections cover those concepts.
* Consumers and Consumer Groups
* Suppose you have an application that needs to read messages from a Kafka topic, run some validations against them, and write the results to another data store. In this case your application will create a consumer object, subscribe to the appropriate topic, and start receiving messages, validating them and writing the results. This may work well for a while, but what if the rate at which producers write messages to the topic exceeds the rate at which your application can validate them? If you are limited to a single consumer reading and processing the data, your application may fall farther and farther behind, unable to keep up with the rate of incoming messages. Obviously there is a need to scale consumption from topics. Just like multiple producers can write to the same topic, we need to allow multiple consumers to read from the same topic, splitting the data between them.
* Kafka consumers are typically part of a consumer group. When multiple consumers are subscribed to a topic and belong to the same consumer group, each consumer in the group will receive messages from a different subset of the partitions in the topic.
* Let’s take topic T1 with four partitions. Now suppose we created a new consumer, C1, which is the only consumer in group G1, and use it to subscribe to topic T1. Consumer C1 will get all messages from all four T1 partitions. See [Figure 4-1](https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html#T1_four_partitions).

­­­­­­­­­­­­­­­­­­­

**Consume the messages from topic using consumer GROUP id:**

Let’s assume that we have four partitions in topic and two consumers in same group and when we run the two consumer instances and partition will be shared among two consumers each consumer have two partitions at this time.

**Note**: We don’t know which partition will go which consumer but it will share among two consumers.

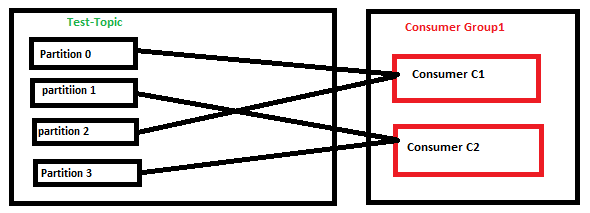
Within same group: **NO**

* Two consumers (*Consumer 1, 2*) within the same group (*Group 1*) **CAN NOT** consume the same message from partition (*Partition 0*).

Across different groups: **YES**

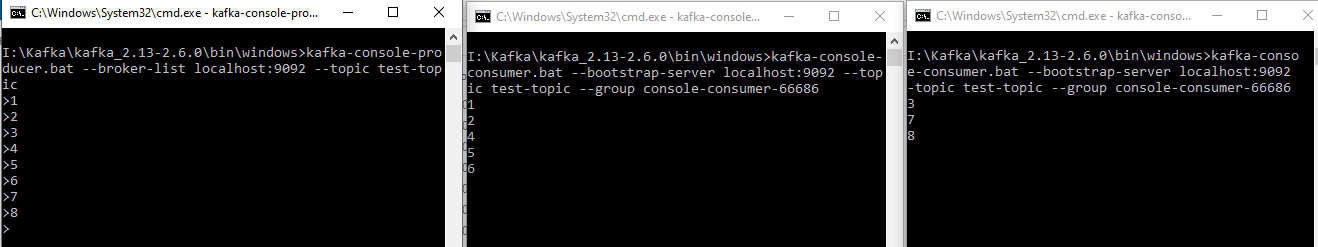
* Two consumers in two groups (*Consumer 1* from *Group 1*, *Consumer 1* from *Group 2*) **CAN** consume the same message from partition (*Partition 0*).

If we add another consumer, C2, to group G1, each consumer will only get messages from two partitions. Perhaps messages from partition 0 and 2 go to C1 and messages from partitions 1 and 3 go to consumer C2.



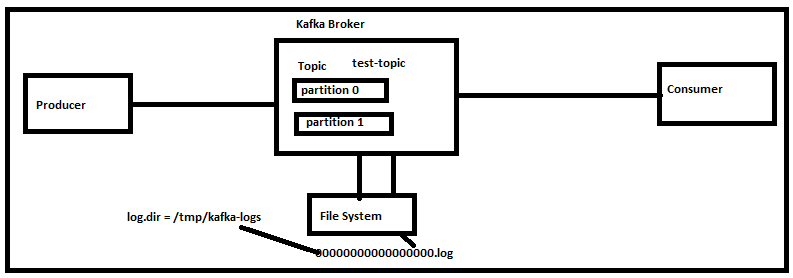
**Note:** In this case two consumers will read the messages in parallel.

**Single Producer** **Consumer1** **Consumer2**



**Commit Log and Retention Policy:**

One of the key concept of retention policy in kafka. That we need to keep the messages in topic for certain amount of period.

****

* As shown in figure above
* When we produce the message **ABC** into topic name **test-topic** it will be first store in the file system as log file **ooooooooooo.log** this log file will store in the below directory **/tmp/kafka-logs** in linux system and in windows where kafka installed **I:\tmp\kafka-logs.** The path can be specified in server. properties file. And consumer will poll the message from topic.
* We have created topic with four partitions and partition will be created like below.

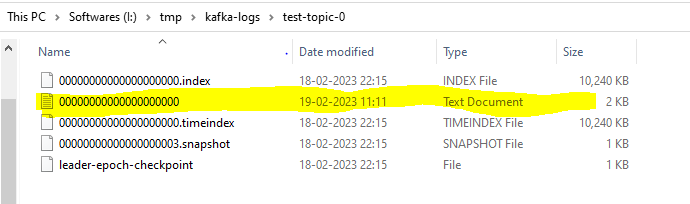
Test-topic-0

Test-topic-1

Test-topic-2

Test-topic-3

* Each partition has unique log file.

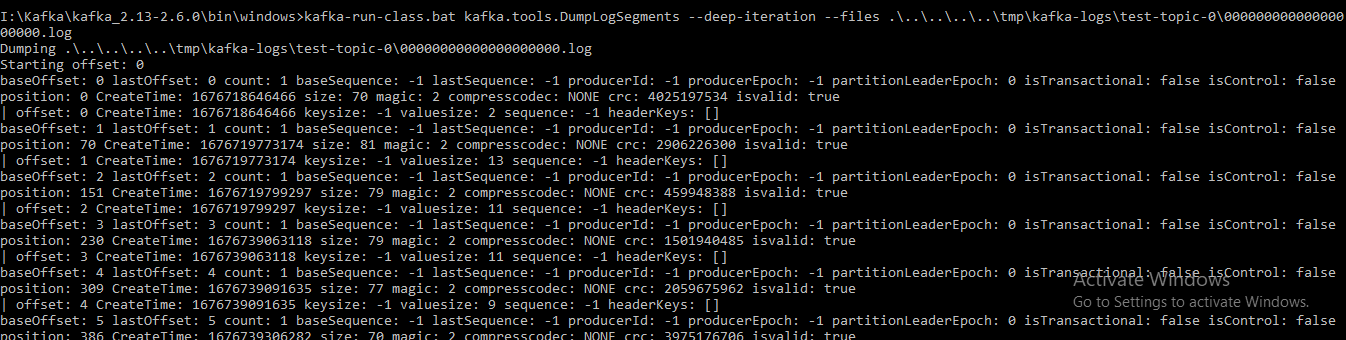
****

Note: Message will be store in the partition log file in the form of binary.

**Viewing the Commit log**

Command: kafka-run-class.bat kafka.tools.DumpLogSegments --deep-iteration --file <partition log file path>

**Ex**: kafka-run-class.bat kafka.tools.DumpLogSegments --deep-iteration --files I:\temp\kafka-logs\test-log-0\000000000000000000000000000.log



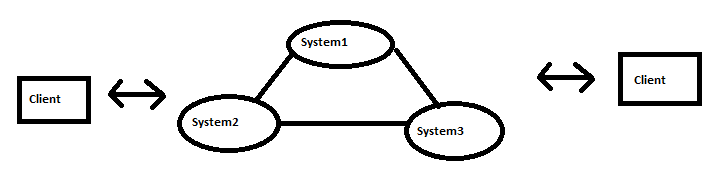
**Note:** Each partition log file has bunch of information like partition offset starting from 0 and createdTime, CompressCode, baseSequence= -1 , lastSequence = -1, producerID = -1 , ProducerEpoch and PartionLeaderEpoch and etc.

**Retention Policy:**

* Determine how long the messages are retained?
* Configured using the property **log.retention.hours=168** in **server.properties** file.
* Default retention policy is **168 hours [7 days]**

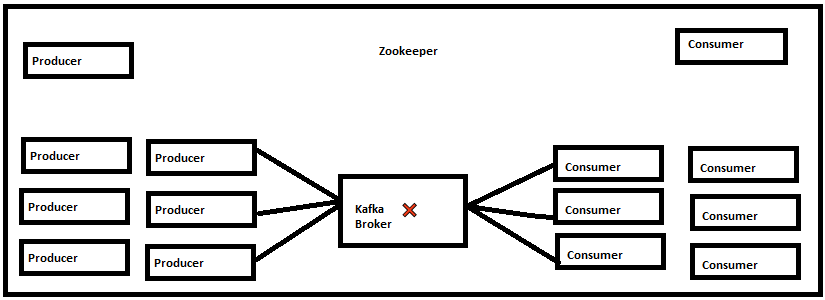
**Kafka as a Distributed Streaming System:**

* Apache Kafka is distributed streaming system.
* Distributed systems are a collections of systems working together to deliver a value.

****

**Some of the key characteristics of the distributed system:**

* Availability and Fault tolerance.
* Reliable work distributions
* Easily Scalable
* Handling concurrency is fairly easy.

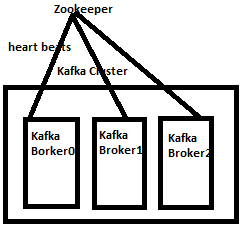
****

**Note:** Let’s have brief discussion about above figure.

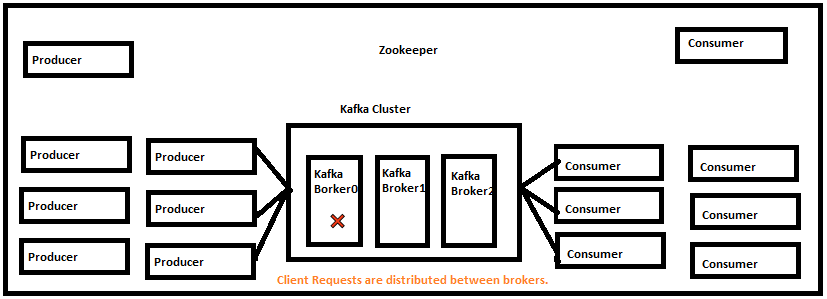
* We have six producers single Kafka Broker and six Consumers.
* When producers start producing the bunch of messages into single Kafka broker and consumers start consuming the records from the same Kafka Broker then Kafka broker may fail or become a poor or down or may crash then there is no way to serve at this time.
* This leads to single point of failure, how to make this better.
* Let me introduce kafka cluster.

**Kafka Clusters:**

* Clusters are typically group of brokers
* Cluster are managed by zookeeper.
* It’s pretty having more than one kafka broker in real time to prevent the data loss.



Kafka Brokers continuously send heart beat to zookeeper at regular interval to ensure the speed of kaka health or active to server client requests.



Note: Here we are having six producers , one kafka cluster and 6 consumer

* When producers start producing the bunch of messages into Kafka brokers and consumers start consuming the same records from the Kafka.
* And some reason first Broker 0 is down or fail.
* We have another two more brokers, those broker will start receiving the message without data loss.
* Client requests are distributed between brokers.
* Easy to scale by adding more brokers based on the need.
* Handles data loss using Replication.

**Setting up Kafka Cluster in Local with 3 Brokers:**

* So far we seen single Kafka Broker
* **Server.properties** file already contain broker.id=0 and running on localhost:9092 as **First Broker**.
* Now let us create two more brokers with following configuration as per our requirement like below.
* Create new Server.poperties file for **second broker** with **Server-1.properties** and have the below configuration.

broker.id=1

listeners=PLAINTEXT://localhost:9093

logs.dirs=/tmp/kafka-logs-1

auto.create.topics.enable=false(optional)

* And create one more server.propeties file for **third broker** with name **Server-2.properties** file

broker.id=2

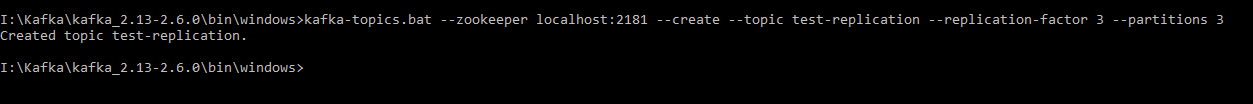
listeners=PLAINTEXT://localhost:9094

logs.dirs=/tmp/kafka-logs-2

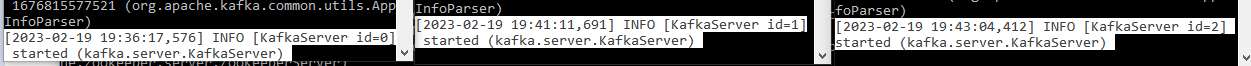
auto.create.topics.enable=false(optional)

* So far we have created two new server properties file as per our requirement.
* Now create the new Topic name **test-replication**
* **Bin\windows\>kafka-topics.bat –create –topic test-replication–zookeeper localhost:2181 –replication-factor 3 –partitions 3**

Created topic test-replication



* Now this is the time start the three kafka brokers by passing topic name and respective server.properties as an argument.



* Look at [Kafka id= 0] started, [Kafka id=1] started and [Kafka id=2] started means all three brokers are started successfully.
* And produce some messages into topic test-cluter.
* And other side start the kafka consumer for consuming the messages from the same topic **test-replication**.

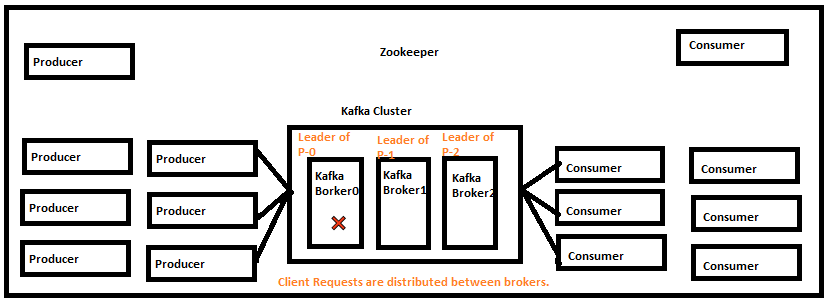
**How Kafka Cluster Distribute the Client Requests?**

* Partition leader is assigned during Topic creation
* Clients will only invoke the leader of the partition to produce or consume data.
* Load is evenly distributed between brokers.

**How Kafka Handles the Data Loss?**

* Kafka handles the data loss using **Replication.**
* Replication Factor number is always having the equal to number of brokers is recommended.
* Replication is set by at the time of topic creation.

kafka-topics.bat –create –topic test-replication –zookeeper locahost:2181 –replication-factor 3 –partitions 3



* When we produce the messages to topic the Zookeeper elect the leader of the broker and the leader will receive the messages and stores in file system as partitions and also it replicates in another brokers bcz we have created replication factor is 3.
* So now let’s assume that Broker-0 is down some reason and the data will be available in Broker-1 and Broker-2 bcz we have created replication factor is 3.
* Now zookeeper identifies that Broker-0 is down and zookeeper is assigned Leader of the Partition is Broker-1 with Leader of Partition 0 and Partition 1.
* So now even though the Leader of the P-0 broker is down the data will be available in another two brokers and consumers will consumer the messages from the topic without data loss.
* This is the overall about Replication Factor.

**In Sync Replica (ISR)**

* In sync replica represents the number of replica in sync with each other in the cluster.
* Included both Leaders and Followers replica
* Recommended value is always greater than 1.
* Ideal value is **ISR = Replication Factor**
* This can be controlled **by min.insync.replicas** property.
* It can be set broker or topic level

**How to check the how many topics and consumer groups are created in kafka**

I:\Kafka\kafka\_2.13-2.6.0\bin\windows>kafka-topics.bat --zookeeper localhost:2181 --list

\_\_consumer\_offsets

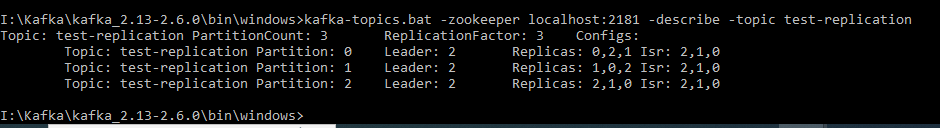
states

test-topic

test-replication

**How to check the how many partitions are created for a topic**

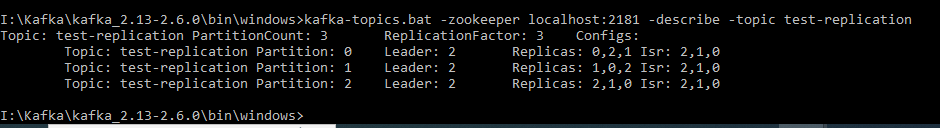
**I:\Kafka\kafka\_2.13-2.6.0\bin\windows>kafka-topics.bat -zookeeper localhost:2181 -describe -topic test-replication**

****

Note: Three partitions are created P -0, P-1 and P-2

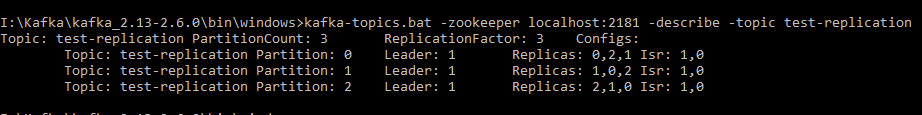
**How to check the who is the Leader of Broker and Replications and Insync**

>kafka-topics.bat –zookeeper localhost:2181 –describe –topic test-topic



* Broker 2 is leader for all three partition
* Partition 0 Replicas are 0 2 1
* Partition 1 Replicas are 1 0 1
* Partition 2 Replicas are 2 1 0
* Broker 0 ISR is 2 1 0
* Broker 1 ISR is 2 1 0
* Broker 2 ISR is 2 1 0

**Note:** Now let’s bring down the broker-2 and again run the above command and compare the both output.

 > Broker 1 is leader for partition 1 and 0

* Partition 0 Replicas are 0 2 1
* Partition 1 Replicas are 1 0 2
* Partition 2 Replicas are 2 1 0
* Broker 0 ISR is 1 0
* Broker 1 ISR is 1 0

**Overview of the application – Library Inventory**

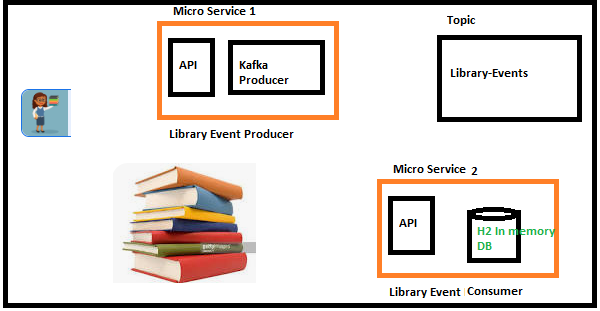
Library Inventory Flow

Source code link is below

<https://github.com/dilipsundarraj1/kafka-for-developers-using-spring-boot>

Spring docs io

https://docs.spring.io/spring-kafka/docs/1.1.1.RELEASE/reference/htmlsingle/#\_kafkatemplate



**Build the Library Event Producer Micro Service:**

**Build Library Event Domain**

package com.kafka.domain;

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Data

public class **EventLibrary**{

private Long eventLibraryId;

private Book book;

}

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Data

public class **Book**{

private Integer bookId;

private String bookName;

private String bookAuthor;

}

**Create the POST endpoint “/libraryEvent”**

@RestController

@RequestMapping(“/v1)

public class LibraryEventController{

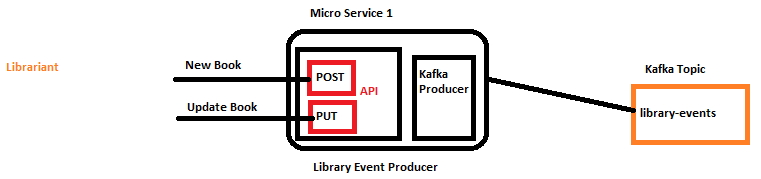
@PostMapping(“/libraryEvent”)

public ResponseEntity<LibraryEvent> libraryEvent(@RequestBody EventLibrary libraryEvent){

//introduce kafka event producer

return ResponseEntity.status(HttpStatus.OK).body(libraryEvent);

}

****

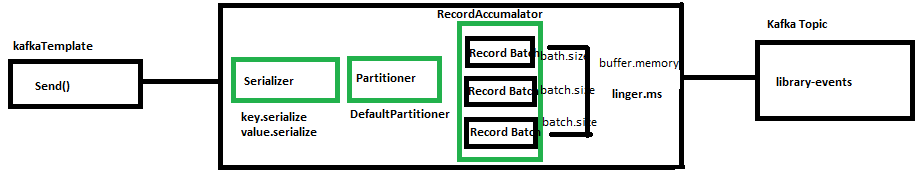
**Introducing to Spring Kafka Template to Produce Messages:**

Link : <https://docs.spring.io/spring-kafka/docs/1.1.1.RELEASE/reference/htmlsingle/#_kafkatemplate>

**KafkaTemplate:**

* Produce messages into kafka topic.
* Similar to JDBCTemplate for DB

**KafkaTemplate.send():**

****

**Note:** When we push the messages intoKafka topic , then it will follow the above steps.

Layer1: **Serializer**: Any records that send to kafka topic, that needs to be serialize.

There are two types of serialize techniques that needs to apply

Key.serializer

Value.serializer

This configuration needs for any producer.

Java KafkaTemplate comes with default serializer we will use them.

Layer2: **Partitioner**:

This layer determines which partition the message go into the topic.

KafkaTemplate Api comes with default partitioner, there are other options to override this configuration too.

Layer3: **RecordAccumlator**:

Any record that sends from kafkaTemplate won’t send kafka topic immediately.

RecordAccumlator buffer the records and sends to the kafka topic once the buffer is full.

The reason is for this approach to limit the number of trips from the application to kafka cluster.

And this eventually avoid the overheads of bombarding with kafka cluster numorous requests. And also helps overall to improving the system performance.

The Record Batch is here partition and topic combination. If we have topic with three partitions then we have three records in the batch. Each and every record batch is the batch size which is represents by the **batch.size** property and the value represents as number of bytes. And also has the overall buffer memory which is represents by the **buffer.memory** property.

**Note**: There is a scenario messages go’s to the Kafka Topic once Batch is the full and also other scenario is there when kafka producer api is not wait for the so long to send the messages to the Kafka Topic.

There is also another handy property called **linger.ms** will be used to send the messages to kafka topic as the value to represents this property in milli seconds.

If the batch records is not full in the record accumulator and is not meet **linger.ms value** and records will be sent to Kafka Topic.

**Configure KafkaTemplate:**

Step1: To configure KafkaTemplate we need the following mandatory configuration

bootstrap-servers: loclahost:9092, localhost:9093, loclahost:9094

key-serializer: org.apache.kafka.common.serialization.IntegerSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

Note: We can also write our own serializer when dealing with complicated objects.

**KafkaTemplate AutoConfiguration:**

**application.yml**

spring:

profiles: local

kafka:

producer:

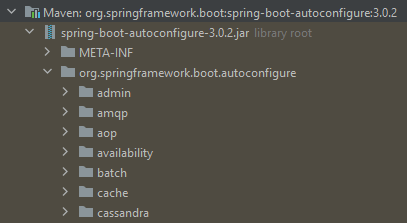
bootstrap-servers: loclahost:9092, localhost:9093, loclahost:9094

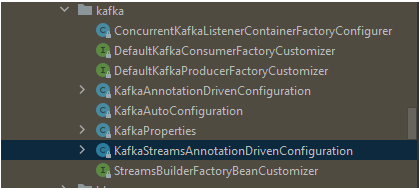
key-serializer: org.apache.kafka.common.serialization.IntegerSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

**How Spring Boot KafkaAutoConfiguration works?**

Go to the project and look to External Dependencies and expand it then search for spring boot autoconfigurer





**Auto Create Topic Using Kafka Admin**

**Note:** I would like to call out it’s not recommended to create topic using Kafka Admin in prod.

**KafkaAdmin:**

* Create Topic using programmatically
* Part of Spring Kafka.
* How to create topic from code point of view?
* Create a Bean of type KafkaAdmin in SpringConfiguration.
* Create a Bean of type NewTopic in SpringConfiguration.

Spring boot version : 3.0.2

package com.kafka.config;  
  
import org.apache.kafka.clients.admin.NewTopic;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Profile;  
import org.springframework.kafka.config.TopicBuilder;  
  
@Configuration  
@Profile("local")  
public class AutoCreateConfig {  
 @Bean  
 public NewTopic libraryEvents() {  
  
 return TopicBuilder.*name*("library-events")  
 .partitions(3)  
 .replicas(3)  
 .build();  
 }  
}

Note: For this we need to configure the Kafka Admin properties in application.yml file.

**Spring**:

Kafka:

Producer:

**Admin**:

Properties:

Bootsrap.server: localhost:9092, localhost:9093, localhost:9094

**Note:** Now this is the time to run the **LibraryEventProducerApplication.java**

**Note**: Make sure zookeeper and all brokers are run before run the application.

Once zookeeper and brokers are run and start the above application and see the whether the topic is created or not for that run the below command.

bin\windows>kafka-topics.bat –zookeeper localhost:2181 --list

**library-events**

**Build Library Events Producer using KafkaTemplate Approach1 (Using Async method is sendDefault(-, -):**

package com.kafka.service;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.kafka.domain.EventLibrary;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.kafka.support.SendResult;  
  
import java.util.concurrent.CompletableFuture;  
  
*/\*\*  
 \** ***@author*** *RAMI REDDY VAKAMALLA  
 \** ***@since*** *22-FEB-2023  
 \*/*@Configuration  
@Slf4j  
public class LibraryEventProducer {  
  
 @Autowired  
 private KafkaTemplate<Long, String> kafkaTemplate;  
  
 @Autowired  
 private ObjectMapper objectMapper;  
  
 public void sendLibraryEvent\_Async(EventLibrary eventLibrary) throws JsonProcessingException {  
 *log*.info("Start producing messages into kafka topic");  
 Long key = eventLibrary.getEventLibraryId();  
 String value = objectMapper.writeValueAsString((eventLibrary);  
 CompletableFuture<SendResult<Long, String>> completableFuture = kafkaTemplate.sendDefault(key, value));  
 completableFuture.whenComplete((result, ex) ->{  
 if(ex == null){  
 handleSuccess(key,value, result);  
 }else{  
 handleError(ex);  
 }  
 });  
 *log*.info("Completed producing messages into kafka topic");  
 }  
  
 private void handleError(Throwable ex) {  
 *log*.error("Error Sending the Message and Exception is {} ", ex.getMessage());  
 try {  
 throw ex;  
 } catch (Throwable e) {  
 *log*.error("Error in onFailure {} ", e.getMessage());  
 }  
 }  
  
 private void handleSuccess(Long key, String value, SendResult<Long, String> result) {  
 *log*.info("Message sent successfully for the Key {} , Value {} and partition is {|",key, value, result.getRecordMetadata().partition());  
 }  
}

**Note**: With this we have successfully build the kafka producer to send the messages to kafka topic.

But in the above code we have not set the kafka-topic. Here i will prefer the application.yml file to specify the topic.

spring:  
 kafka:  
 producer:  
 bootstrap-servers: localhost:9092, localhost:9093, localhost:9094  
 key-serializer: org.apache.kafka.common.serialization.LongSerializer  
 value-serializer: org.apache.kafka.common.serialization.StringSerializer  
 admin:  
 properties:  
 bootsrap.server: localhost:9092, localhost:9093, localhost:9094  
 template:  
 default-topic: library-events

**Note**: With this we have successfully completed building the Kafka Producer

**Build the Event Producer Controller Layer for Kafka Producer:**

package com.kafka.controller;

/\*\*

\* @author RAMI REDDY VAKAMALLA

\* @since 22-FEB-2023

\*/  
@RestController  
@RequestMapping("/v1")  
public class EventLibraryProducerController {  
  
 @Autowired  
 private LibraryEventProducer libraryEventProducer;  
  
 @PostMapping("/libraryevent")  
 public ResponseEntity<EventLibrary> processEventLibrary(@RequestBody EventLibrary eventLibrary) throws JsonProcessingException {  
 //invoke library event producer  
 libraryEventProducer.sendLibraryEvent\_Async(eventLibrary);  
 return ResponseEntity.status(HttpStatus.CREATED).body(eventLibrary);  
 }  
}

**Note:** with this we have successfully build EventLibraryProducerController class.

**Note:** Now let’s run the spring boot application and send the event library event message as json format and hit to the backend application using postman.

But be sure that zookeeper and all three brokers are up and running as we said before running the application else you will see connection refused errors in console so try to avoid this and make sure.

**Note: Once application runs successfully, you will see the following information console.**

2023-02-22T06:54:56.318+05:30 INFO 8672 --- [ main] c.kafka.LibraryEventProducerApplication : Starting LibraryEventProducerApplication using Java 19 with PID 8672 (I:\Kafka\Kafka Projects\library-event-producer\target\classes started by LENOVO in I:\Kafka\Kafka Projects\library-event-producer)

2023-02-22T06:54:56.330+05:30 INFO 8672 --- [ main] c.kafka.LibraryEventProducerApplication : **The following 1 profile is active: "local"**

2023-02-22T06:55:00.384+05:30 INFO 8672 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : **Tomcat initialized with port(s): 8090 (http)**

2023-02-22T06:55:00.425+05:30 INFO 8672 --- [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2023-02-22T06:55:00.426+05:30 INFO 8672 --- [ main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.5]

2023-02-22T06:55:00.968+05:30 INFO 8672 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2023-02-22T06:55:00.983+05:30 INFO 8672 --- [ main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4460 ms

2023-02-22T06:55:01.893+05:30 INFO 8672 --- [ main] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page: class path resource [static/index.html]

2023-02-22T06:55:02.873+05:30 INFO 8672 --- [ main] **o.a.k.clients.admin.AdminClientConfig : AdminClientConfig values:**

**bootstrap.servers = [localhost:9092]**

client.dns.lookup = use\_all\_dns\_ips

client.id =

connections.max.idle.ms = 300000

default.api.timeout.ms = 60000

metadata.max.age.ms = 300000

metric.reporters = []

metrics.num.samples = 2

metrics.recording.level = INFO

metrics.sample.window.ms = 30000

receive.buffer.bytes = 65536

reconnect.backoff.max.ms = 1000

reconnect.backoff.ms = 50

request.timeout.ms = 30000

retries = 2147483647

retry.backoff.ms = 100

//here am cutting down many sasl and sslproperties

ssl.truststore.type = JKS

2023-02-22T06:55:03.163+05:30 WARN 8672 --- [ main**] o.a.k.clients.admin.AdminClientConfig : These configurations '[bootsrap.server]' were supplied but are not used yet.**

2023-02-22T06:55:03.200+05:30 INFO 8672 --- [ main] o.a.kafka.common.utils.AppInfoParser : Kafka version: 3.3.2

2023-02-22T06:55:03.201+05:30 INFO 8672 --- [ main] o.a.kafka.common.utils.AppInfoParser : Kafka commitId: b66af662e61082cb

2023-02-22T06:55:03.201+05:30 INFO 8672 --- [ main] o.a.kafka.common.utils.AppInfoParser : Kafka startTimeMs: 1677029103169

2023-02-22T06:55:05.347+05:30 INFO 8672 --- [| adminclient-1] o.a.kafka.common.utils.AppInfoParser : App info kafka.admin.client for adminclient-1 unregistered

2023-02-22T06:55:05.436+05:30 INFO 8672 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8090 (http) with context path ''

2023-02-22T06:55:05.472+05:30 INFO 8672 --- [ main] c.kafka.LibraryEventProducerApplication : Started LibraryEventProducerApplication in 10.868 seconds (process running for 15.147)

2023-02-22T09:12:45.915+05:30 INFO 8672 --- [nio-8090-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'

2023-02-22T09:12:45.938+05:30 INFO 8672 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'

2023-02-22T09:12:46.028+05:30 INFO 8672 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 89 ms

URL**:** [**http://localhost:8090/kafka/spring/v1/libraryevent**](http://localhost:8090/kafka/spring/v1/libraryevent)

**Request:** { "eventLibraryId": 1,

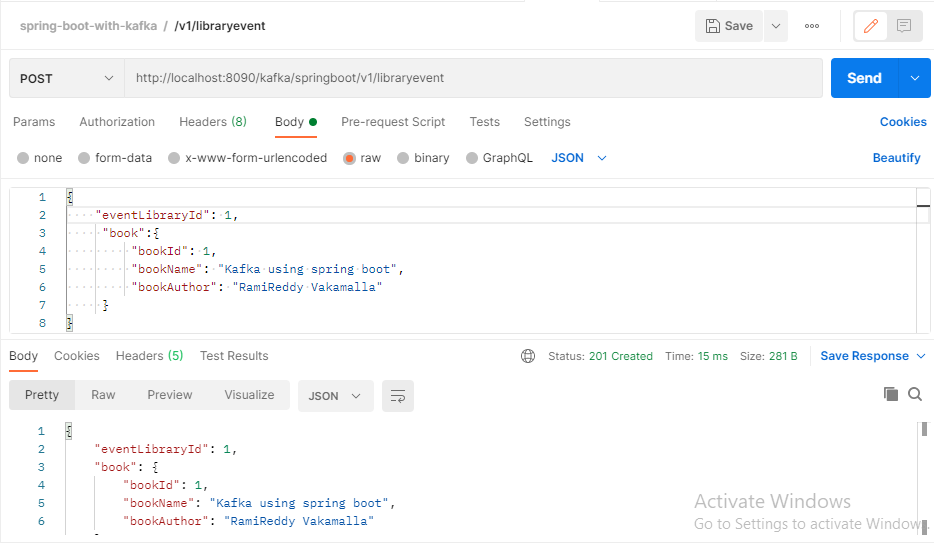
"book": {

"bookId": 1,

"bookName": "Kafka using spring boot",

"bookAuthor": "RamiReddy Vakamalla"

}}



**Note:** After hitting the very first request you will see the below logs in console as **ProducerConfig** values.

2023-02-22T10:02:48.397+05:30 INFO 7752 --- [nio-8090-exec-1] o.a.k.clients.producer.ProducerConfig : ProducerConfig values:

acks = -1

batch.size = 16384

**bootstrap.servers = [localhost:9092, localhost:9093, localhost:9094]**

buffer.memory = 33554432

client.dns.lookup = use\_all\_dns\_ips

client.id = producer-1

compression.type = none

connections.max.idle.ms = 540000

delivery.timeout.ms = 120000

enable.idempotence = true

interceptor.classes = []

**key.serializer = class org.apache.kafka.common.serialization.LongSerializer**

**linger.ms = 0**

max.block.ms = 60000

max.in.flight.requests.per.connection = 5

max.request.size = 1048576

metadata.max.age.ms = 300000

metadata.max.idle.ms = 300000

metric.reporters = []

metrics.num.samples = 2

metrics.recording.level = INFO

metrics.sample.window.ms = 30000

partitioner.adaptive.partitioning.enable = true

partitioner.availability.timeout.ms = 0

partitioner.class = null

partitioner.ignore.keys = false

receive.buffer.bytes = 32768

reconnect.backoff.max.ms = 1000

reconnect.backoff.ms = 50

request.timeout.ms = 30000

retries = 2147483647

retry.backoff.ms = 100

//here am cutting down many properties as

transaction.timeout.ms = 60000

transactional.id = null

value.serializer = class org.apache.kafka.common.serialization.StringSerializer

2023-02-22T10:02:48.791+05:30 INFO 7752 --- [nio-8090-exec-1] o.a.kafka.common.utils.AppInfoParser : Kafka version: 3.3.2

2023-02-22T10:02:48.791+05:30 INFO 7752 --- [nio-8090-exec-1] o.a.kafka.common.utils.AppInfoParser : Kafka commitId: b66af662e61082cb

2023-02-22T10:02:48.791+05:30 INFO 7752 --- [nio-8090-exec-1] o.a.kafka.common.utils.AppInfoParser : Kafka startTimeMs: 1677040368791

2023-02-22T10:02:49.106+05:30 INFO 7752 --- [ad | producer-1] org.apache.kafka.clients.Metadata : [Producer clientId=producer-1] Cluster ID: laT5czkvQwa6W7WRu8OaVg

2023-02-22T10:02:49.159+05:30 INFO 7752 --- [ad | producer-1] o.a.k.c.p.internals.TransactionManager : [Producer clientId=producer-1] ProducerId set to 14001 with epoch 0

2023-02-22T10:02:49.206+05:30 INFO 7752 --- [nio-8090-exec-1] com.kafka.service.LibraryEventProducer : Completed producing messages into kafka topic

2023-02-22T10:02:50.206+05:30 INFO 7752 --- [ad | producer-1] com.kafka.service.LibraryEventProducer : **Message sent successfully for the Key 1 , Value {"eventLibraryId":1,"book":{"bookId":1,"bookName":"Kafka using spring boot","bookAuthor":"RamiReddy Vakamalla"}}** and partition is {|

2023-02-22T10:02:52.789+05:30 INFO 7752 --- [nio-8090-exec-3] com.kafka.service.LibraryEventProducer : Start producing messages into kafka topic

2023-02-22T10:02:52.796+05:30 INFO 7752 --- [nio-8090-exec-3] com.kafka.service.LibraryEventProducer : Completed producing messages into kafka topic

2023-02-22T10:53:17.850+05:30 INFO 7752 --- [ad | producer-1] com.kafka.service.LibraryEventProducer : Message sent successfully for the Key null , Value {"eventLibraryId":null,"book":{"bookId":1,"bookName":"Kafka using spring boot","bookAuthor":"RamiReddy Vakamalla"}} and partition is {|

**Note1**: When we push the message **key** as null, then kafka uses Default Partition.

**Note2:** If you observe above highlighted text along with red colour text, Here are we are seeing log message as Completed producing messages into kafka topic, before printing the sent successfully for the Key. This because **kafkaTemplate.sendDefault()** methods runs in ASYNCRONOUSLY.

**Note3:** If we want to run send the messages into kafka topic in SYNCRONOUSLY then call the **get()** overloaded method on sendDefault() method.

**Build Library Events Producer using KafkaTemplate Approach2 using sendDefault(-, -) Synchronously:**

public void sendLibraryEventsInSynchrounously(EventLibrary eventLibrary) throws JsonProcessingException, ExecutionException, InterruptedException {  
 *log*.info("Started producing messages into kafka topic in synchronously! ");  
 Long key = eventLibrary.getEventLibraryId();  
 String value = objectMapper.writeValueAsString(eventLibrary);  
 SendResult<Long, String> sendResult = kafkaTemplate.sendDefault(key, value).get();  
 //here get() is the overloaded method, we can also specify the timeout value, in TimeUnit.seconds or milliseconds like get(1, TimeUnits.SECONDS)  
 *log*.info("send result response {} ", sendResult.toString());  
 *log*.info("Completed producing messages into kafka topic in synchronously! ");  
}

**Note**: Just call above method in your above controller while commenting the async method and see the below synchronous result.

2023-02-22T11:22:53.389+05:30 INFO 9456 --- [nio-8090-exec-8] com.kafka.service.LibraryEventProducer : Started producing messages into kafka topic in synchronously!

2023-02-22T11:22:53.574+05:30 INFO 9456 --- [nio-8090-exec-8] o.a.k.clients.producer.ProducerConfig : ProducerConfig values:

acks = -1

batch.size = 16384

bootstrap.servers = [localhost:9092, localhost:9093, localhost:9094]

2023-02-22T11:22:56.568+05:30 INFO 9456 --- [nio-8090-exec-8] com.kafka.service.LibraryEventProducer : send result response SendResult [producerRecord=ProducerRecord(topic=library-events, partition=null, headers=RecordHeaders(headers = [], isReadOnly = true), key=null, value={"eventLibraryId":null,"book":{"bookId":1,"bookName":"Kafka using spring boot","bookAuthor":"RamiReddy Vakamalla"}}, timestamp=null), recordMetadata=library-events-0@57]

2023-02-22T11:22:56.568+05:30 INFO 9456 --- [nio-8090-exec-8] com.kafka.service.LibraryEventProducer : Completed producing messages into kafka topic in synchronously!

**Build Library Events Producer using KafkaTemplate Approach3 using send() overloaded method with ProducerRecord including headers pass:**

//Approach3\_1  
public void sendLibraryEvents\_Send\_Async\_with\_ProducerRecord(EventLibrary eventLibrary) throws JsonProcessingException, ExecutionException, InterruptedException {  
 *log*.info("Started producing messages into kafka topic in using send with ProducerRecord! ");  
 Long key = eventLibrary.getEventLibraryId();  
 String value = objectMapper.writeValueAsString(eventLibrary);  
 CompletableFuture<SendResult<Long, String>> sendResult = kafkaTemplate.send(this.buildProducerRecord(key, value));  
 //here get() is the overloaded method, we can also specify the timeout value, in TimeUnit.seconds or milliseconds  
 *log*.info("send result response {} ", sendResult.toString());  
 *log*.info("Completed producing messages into kafka topic using send with ProducerRecord! ");  
}  
  
private ProducerRecord<Long, String> buildProducerRecord(Long key, String value) {  
 //topic, partition, key, value  
 //this event-source is just dummy key and value passing , just for example, like this we can pass in realtime  
 List<Header> headers = List.*of*(new RecordHeader("event-source", "scanner".getBytes()));  
 return new ProducerRecord<>("library-events",null, key, value, headers);  
}

**Note:**  When we use sendDefault() method to send the messages to kafka topic , then it look for the application.yml file for default topic. Example below.

spring:  
 kafka:  
 template:  
 default-topic: library-events

**Writing Juint Test cases for above Code:**

**Why Automated test cases?**

* Manual testing is time consume
* Manual testing slows down the development
* Adding new changes are error prone.

**What are automated test cases?**

* Automated test runs against your code base.
* Automated test runs as part of the build.’
* This is a requirement for today’s software development.
* Easy to capture the bugs.
* Types of automated tests
* Junit
* Integration tests
* End to End tests

**Tools for Automated Tests:**

Junit and spoc

**What are the integration tests?**

Test combines the different layers of code and verifies the behaviour is working as expected.

//Approach1 when dealing with Realtime Brokers, that means when brokers are up and running  
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*)  
class EventProducerControllerTest {  
  
 @Autowired  
 private TestRestTemplate restTemplate;  
  
 @Test  
 void testEventProducer() {  
 Book book = Book.*builder*().bookId(1).bookName("Rami").bookAuthor("Rami Reddy").build();  
 EventLibrary eventLibrary = EventLibrary.*builder*().eventLibraryId(null).book(book).build();  
 HttpHeaders header = new HttpHeaders();  
 header.setContentType(MediaType.*APPLICATION\_JSON*);  
 HttpEntity entity = new HttpEntity(eventLibrary, header);  
 ResponseEntity responseEntity = this.restTemplate.exchange("/v1/libraryevent", HttpMethod.*POST*, entity, EventLibrary.class);  
 Assertions.*assertEquals*(HttpStatus.*CREATED*, responseEntity.getStatusCode());  
 }  
}

Note: Make sure Zookeeper and brokers are up and running before run the test case, bcz we are testing with real time brokers. But this approach is not at all recommended approach, we have to use In memory kafka.

**What is EmbededKafka?**

In memory Kafka

EmbededKafka mainly works in integration test, bcz to make sure our service is fine when we don’t need real time brokers available so this case EmbededKafka will hep us.

Easy to write integration test without using realtime brokers it uses in memory kafka.

//Recommended Approach, bcz test should run on mocking  
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*)  
@EmbeddedKafka(topics = {"library-events"}, partitions = 3)  
@TestPropertySource(properties = {"spring.kafka.admin.properties.bootstrap.servers=${spring.embedded.kafka.brokers}",  
 "spring.kafka.producer.bootstrap.servers=${spring.embedded.kafka.brokers}"})  
class EventProducerControllerTest\_Approach2 {  
  
 @Autowired  
 private TestRestTemplate restTemplate;  
  
 @Autowired  
 private EmbeddedKafkaBroker embeddedKafkaBroker;  
  
  
 private Consumer<Long, String> consumer;  
  
 //creating mock consumer  
 @BeforeEach  
 void setup() {  
 Map<String, Object> config = new HashMap<String, Object>(KafkaTestUtils.*consumerProps*("group1", "true", embeddedKafkaBroker));  
 this.consumer = new DefaultKafkaConsumerFactory<Long, String>(config, LongDeserializer::new, StringDeserializer::new).createConsumer();  
 embeddedKafkaBroker.consumeFromAllEmbeddedTopics(consumer);  
 }  
  
 @Test  
 void testEventProducer\_With\_EmbeddedKafka()throws InterruptedException{  
 Book book = Book.*builder*().bookId(1).bookName("Kafka using spring boot").bookAuthor("RamiReddy Vakamalla").build();  
 EventLibrary eventLibrary = EventLibrary.*builder*().eventLibraryId(null).book(book).build();  
 HttpHeaders header = new HttpHeaders();  
 header.setContentType(MediaType.*APPLICATION\_JSON*);  
 HttpEntity entity = new HttpEntity(eventLibrary, header);  
 ResponseEntity responseEntity = this.restTemplate.exchange("/v1/libraryevent", HttpMethod.*POST*, entity, EventLibrary.class);  
 Assertions.*assertEquals*(HttpStatus.*CREATED*, responseEntity.getStatusCode());  
 ConsumerRecord<Long, String> consumerRecord = KafkaTestUtils.*getSingleRecord*(this.consumer, "library-events");

Thread.sleep(3000);  
 String expectedRecord = "{\n" +  
 " \"eventLibraryId\": null,\n" +  
 " \"book\": {\n" +  
 " \"bookId\": 1,\n" +  
 " \"bookName\": \"Kafka using spring boot\",\n" +  
 " \"bookAuthor\": \"RamiReddy Vakamalla\"\n" +  
 " }\n" +  
 "}";  
 String value = consumerRecord.value();  
 Assertions.*assertEquals*(expectedRecord, value);  
 }  
}

**Junit5 Test case:**

**Update the Events to Kafka Topic:**

**Note:** Kafka guarantees the ordering only at partition level using the keys.

Method: POST

URL: <http://loclhost:8080/kafka/springboot/libraryevent>

Body : {

    "eventLibraryId": 123,

    "book": {

        "bookId": 1,

        "bookName": "Kafka using spring boot",

        "bookAuthor": "RamiReddy Vakamalla"

    }

}

Public Class EventLibraryProducerController{

@PutMapping("/libraryevent")  
public ResponseEntity<?> updateEventLibrary(@RequestBody @Valid EventLibrary eventLibrary) throws JsonProcessingException, ExecutionException, InterruptedException {  
 //invoke library event producer  
 if(eventLibrary.getEventLibraryId() == null){  
 return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("Event Library Id must not be null!");  
 }  
 libraryEventProducer.sendLibraryEvent\_Asycn(eventLibrary);  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(eventLibrary);  
}

}

**EventProducerEvent.java**

//Approach1  
public void sendLibraryEvent\_Async(EventLibrary eventLibrary) throws JsonProcessingException {  
 *log*.info("Start producing messages into kafka topic");  
 Long key = eventLibrary.getEventLibraryId();  
 String value = objectMapper.writeValueAsString(eventLibrary);  
 CompletableFuture<SendResult<Long, String>> completableFuture = kafkaTemplate.sendDefault(key, value);  
 completableFuture.whenComplete((result, ex) ->{  
 if(ex == null){  
 handleSuccess(key,value, result);  
 }else{  
 handleError(ex);  
 }  
 });  
 *log*.info("Completed producing messages into kafka topic");  
}

private void handleError(Throwable ex) {  
 *log*.error("Error Sending the Message and Exception is {} ", ex.getMessage());  
 try {  
 throw ex;  
 } catch (Throwable e) {  
 *log*.error("Error in onFailure {} ", e.getMessage());  
 }  
}  
  
private void handleSuccess(Long key, String value, SendResult<Long, String> result) {  
 *log*.info("Message sent successfully for the Key {} , Value {} and partition is {|",key, value, result.getRecordMetadata().partition());  
}

**Note:** Pls be sure that kafka cluster is up and running before run the application.

**Note**: While you start updating the messages into kafka topic, the key must not be null and it uses the key and perform the ordering using the partition and update the messages in kafka topic.

**Note**: see the below logs it uses same partition while updating[n number of time] the messages into kafka topic.

Message sent successfully for the Key 123,value {some json}, partition is 1

Message sent successfully for the Key 123,value {some json}, partition is 1

**Note**: If you send **libraryEventId** as null then it uses the default partition.

//Assignment1: Integration test for PUT /libraryEvent using EmbeddedKafka and

//Assignment2: Junit test case for PUT /libraryEvent using Junit5 mockito

**Kafka Producer Important Configurations:**

* **Acks**
* Acks = 0, 1 and all
* Acks = 0 -> No guarantees (Not recommended) the messages written into leader replica or follower replica.
* Acks = 1 -> guarantees the message is written into leader replica (default ) [recommended]
* Acks = all -> guarantees the message is written into leader and all all the replicas [recommended
* **Retries**
* Integer value is : [0 to 2147483647]
* In spring kafka default value is : **2147483647**
* **Retry.backoff.ms**
* Integer value represents in milliseconds
* Default value is 100ms.

**Note**: we can override the producer config values in application.yml file: like acks from 1 to all and etc..

spring:  
 kafka:  
 producer:  
 bootstrap-servers: localhost:9092, localhost:9093, localhost:9094  
 key-serializer: org.apache.kafka.common.serialization.LongSerializer  
 value-serializer: org.apache.kafka.common.serialization.StringSerializer  
 properties:   
 acks: ALL  
 retries: 10