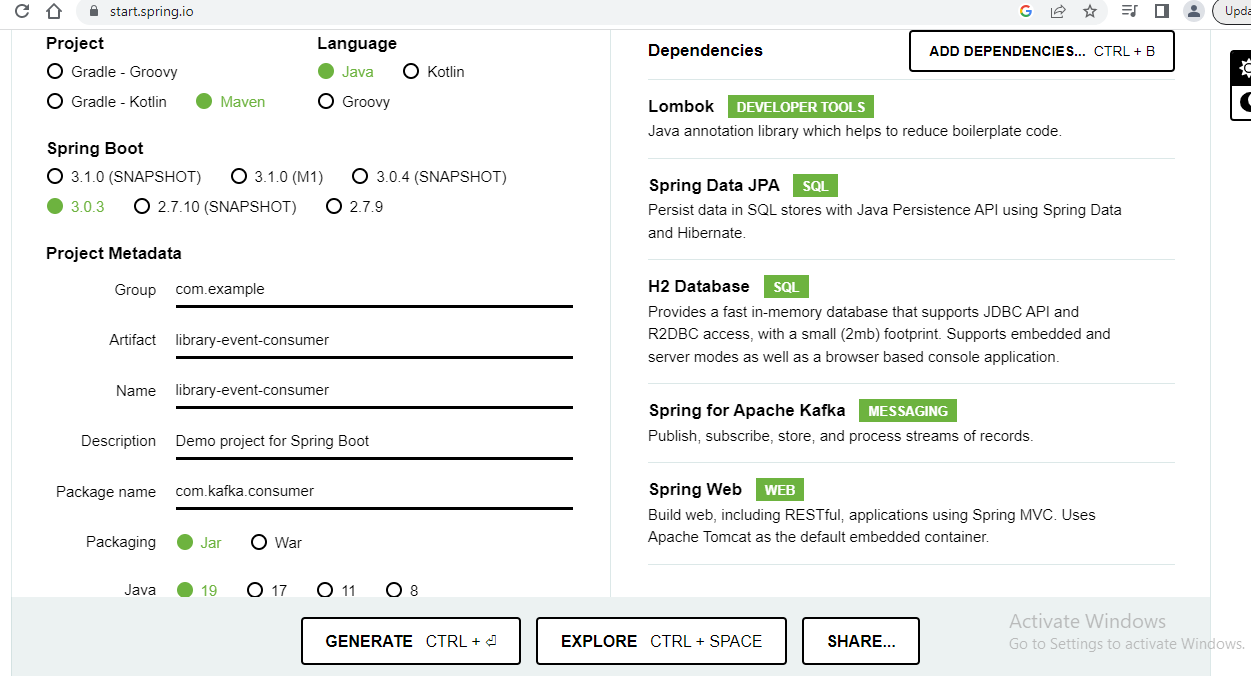
**Apache Kafka Consumer Base Project:**

**Note:** Please refer the below link for consume the records from the topic.

<https://docs.spring.io/spring-kafka/reference/html/#receiving-messages> [4.1.4]

**Step1**: In order to create new spring boot project go to below link and provide the group id, artefact id , name and add the necessary dependencies and generate the project.

URL : <https://start.spring.io/>

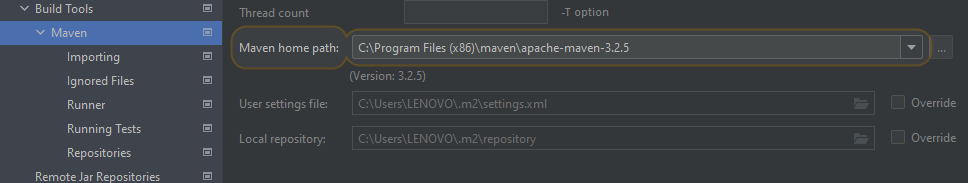


**Step2**: Once project is downloaded then unzip it and import to eclipse or intellij.

**Step3:** Once project is imported into intellij then add provide the maven installation path.

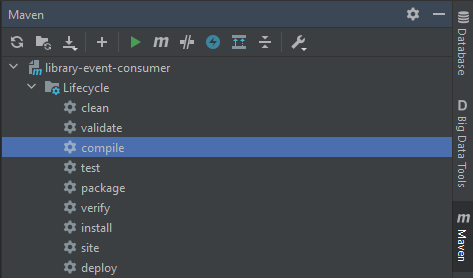
For this follow below

cntrl+alt and press s then it will open the window and search for maven and give the maven path. C:/Program Files (x86)/maven/apache-maven-3.2.5



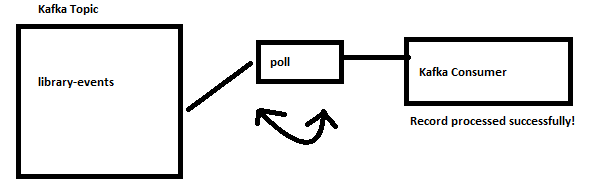
**Step4**: once maven path is added then click apply and ok button.

**Step5:** Now do the maven clean and compile.



* It will download the all necessary dependencies what we have added in pom.xml file.

**Step6:** Read the messages from library-events kafka topic using Kafka Consumer.

****

**Spring Kafka Consumer:**

**Approach1:**

* **MessageListenerContainer**

KafkaMessageListenerContainer:

* Implementation of MessageListenerContainer
* Polls the record from the kafka topic
* Commits the offset automatically
* Uses single threaded

ConcurrentMessageListenerContainer

* Represents multiple KafkaMessageListenerContainer.
* We can spin up multiple instance of message listener and it polls the messages from kafka topic , same approach will not work in KafkaMessageListenerContainer.

**Appproach2:**

* **@**KafkaListener annotation

It uses ConcurrentMessageListenerContainer behind the scene.

* This is the simplest way to build the Kafka Consumer.
* KafkaListener sample code.

**4.1.4. Receiving Messages**

You can receive messages by configuring a MessageListenerContainer and providing a message listener or by using the @KafkaListener annotation.

**Message Listeners**

When you use a [message listener container](https://docs.spring.io/spring-kafka/reference/html/#message-listener-container), you must provide a listener to receive data. There are currently eight supported interfaces for message listeners. The following listing shows these interfaces:

**Note**: read the link https://docs.spring.io/spring-kafka/reference/html/#receiving-messages

@**KafkaListener** annotation is used to designate a bean method as a listener for a listener container. The bean is wrapped in a **MessagingMessageListenerAdapter** configured with various features, such as converters to convert the data, if necessary, to match the method parameters.

You can configure most attributes on the annotation with SpEL by using #{…​} or property placeholders (${…​}). See the [Javadoc](https://docs.spring.io/spring-kafka/api/org/springframework/kafka/annotation/KafkaListener.html) for more information

@Configuration  
@Slf4j  
public class LibraryEventConsumer {  
 //@link https://docs.spring.io/spring-kafka/reference/html/#receiving-messages  
 @KafkaListener(topics = {"library-events"})  
 public void onMessage(ConsumerRecord<Long, String> consumerRecord) {  
 *log*.info("Consume messages from Kafka Topic {} ", consumerRecord);  
 }  
}

This mechanism requires an @**EnableKafka** annotation on one of your @**Configuration** classes and a listener container factory, which is used to configure the underlying  ConcurrentMessageListenerContainer. By default, a bean with name **kafkaListenerContainerFactory** is expected. The following example shows how to use ConcurrentMessageListenerContainer:

@Configuration  
@EnableKafka  
public class LibraryEventsConsumerConfig {

}

**application-local.yml**

spring:  
 kafka:  
 consumer:  
 bootstrap-servers: localhost:9092, localhost:9093, localhost:9094  
 key-deserializer: org.apache.kafka.common.serialization.LongDeserializer  
 value-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 group-id: library-events-listener-group

**Note:** Please make sure that zookeeper and kafka clusters are rup and running before producer and consumer apps are running.

Now publish the messages into kafka topic using postman and other end see the consumer app console you would receive the messages fromkafka topic as well.

**Note1**: So far we have done when both producer and consumer apps up and running and messages sends to kafka topic and consume the messages from kafka topic.

**Note2**: But we have not done yet when consumer is down some reason and messages are still coming to kafka topic, but those messages are not read [from where we left] after consumer is up and running.

But we are able to consume when messages are coming topic after consumer is up only.

**This scenario will see later**

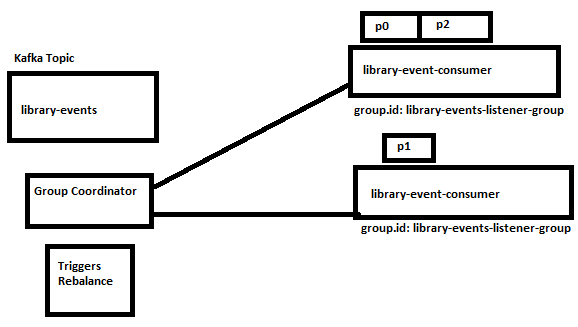
**Consumer Groups and Rebalance:**

**Note1:** So far we had only one consumer to consume the messages from kafka topic, this case consumer group will assign all partition to single consumer only.

Suppose we have these partition p0, p1 and p2 all these are assigned to single consumer only.

**Note2:** Now assume that we have multiple instances of consumers this case how partition will assigned to consumers let see now below.

1. Changing the partition ownership from one consumer to another.



**Note**: here we have two three partitions and two consumer instance, this case group-co-ordinator will check that oh i have two consumers, then triggers rebalance will rebalancing the partitions between two consumers like above.

**Note**: For practice purpose what we will now, we will prepare the jar file for the consumer api and run the both jars in different ports and you can see how partitions are distributed between two instances.

This case group co coordinator will take care completely

Java –jar –Dserver.port = 8082 <xyz.jar>

**Default Committing offset Management in Spring Kafka :**

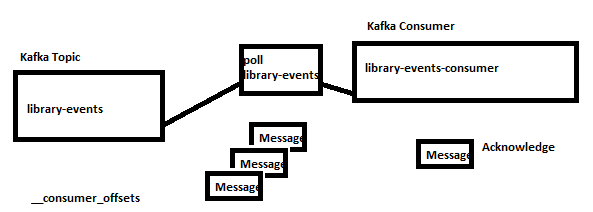
**BATCH** is the default commit offset in spring kafka.

When consumer poll is poll the records from batch then poll commit the offset once batch is free.

Recommended consumer offset is **BATCH.**

**Manual Committing offset:**

How to manage the committing offset manually in application itself.



1. Here once consumer polls the records from topic, then application process the record and application needs to be **Acknowledge** on each record level.
2. Once application process each record and acknowledge then commit the offset.
3. For this we need to override the default configuration.

Package com.kafka.config;

@Configuration  
@EnableKafka  
public class LibraryEventsConsumerConfig {  
  
  
 @Bean  
 ConcurrentKafkaListenerContainerFactory<Long, String>  
 kafkaListenerContainerFactory(ConsumerFactory<Long, String> consumerFactory) {  
 ConcurrentKafkaListenerContainerFactory<Long, String> factory =  
 new ConcurrentKafkaListenerContainerFactory<>();  
 factory.setConsumerFactory(consumerFactory);  
 factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.*MANUAL*);  
 return factory;  
 }  
  
}

1. Next implement the AcknowledgingMessageListener in LibraryEventConsumer Service.

And call the acknowledgement.acknowledge()

1. We can also set **factory.setConcurrency(3),** this means it will create three instances for faster retrieval. But this is not recommended to use in cloud environment.

@Configuration  
@Slf4j  
class LibraryEventConsumerManualCommmitOffset implements AcknowledgingMessageListener<Long, String>{  
  
 @Override  
 public void onMessage(ConsumerRecord<Long, String> consumerRecord, Acknowledgment acknowledgment) {  
 *log*.info("Consume messages from Kafka Topic {} ", consumerRecord);  
 acknowledgment.acknowledge();  
 }  
}

1. Once record is process is successfully by the application we are manually acknowledging that for manual commit offset.

**Custom Error Handler and Custom Retry in Spring Kafka Consumer:**

[**https://docs.spring.io/spring-kafka/reference/html/#annotation-error-handling**](https://docs.spring.io/spring-kafka/reference/html/#annotation-error-handling)

When we encounter with exception during runtime of consume the records from kafka topic, in this case we need to handle the exception and we need to perform retry if required for this do the below code.

@Configuration  
@EnableKafka  
@Slf4j  
public class LibraryEventsConsumerConfig {  
  
 private DefaultErrorHandler defaultErrorHandler() {  
 //when exception occurred it will perform retry with 2 times of every 1 sec  
 var fixedBackoff = new FixedBackOff(1000l, 2);  
 var errorHandler = new DefaultErrorHandler(fixedBackoff);  
 //when we want to monitor the exception do the below code  
 errorHandler.setRetryListeners((record, ex, deliveryAttempts) -> {  
 *log*.error("Failed Record in Retry Listener {}, Exception : {} and deliveryAttempts : {} ", record, ex, deliveryAttempts);  
 });  
 //when we want to ignore the exception to be retry in this case we can add below code.  
 var exceptionToIgnoreList = List.*of*(IllegalArgumentException.class);  
 exceptionToIgnoreList.forEach(errorHandler::addNotRetryableExceptions);  
 return errorHandler;  
 }  
  
 @Bean  
 ConcurrentKafkaListenerContainerFactory<Long, String>  
 kafkaListenerContainerFactory(ConsumerFactory<Long, String> consumerFactory) {  
 ConcurrentKafkaListenerContainerFactory<Long, String> factory =  
 new ConcurrentKafkaListenerContainerFactory<>();  
 factory.setConsumerFactory(consumerFactory);  
 factory.setConcurrency(3); //it's not recommended to use in cloud deployment  
 factory.setCommonErrorHandler(defaultErrorHandler()); //for catching the exception  
 factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.*MANUAL*); //setting manually  
 return factory;  
 }

}

**Recovery Types:**

Approach1: Reprocess the failed records again

Example: Service the consumer interacts with is temporary down.

Approach2: Discard the message and move on

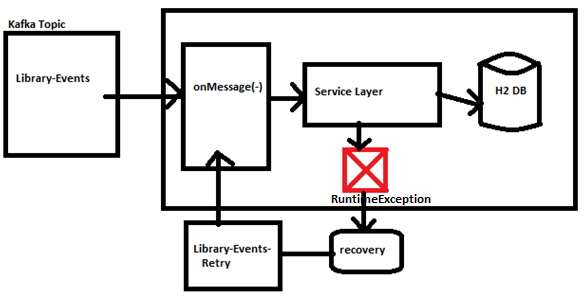
Example: Invalid message, Parsing error and Invalid Event.

**Approach1:** Reprocess the message again

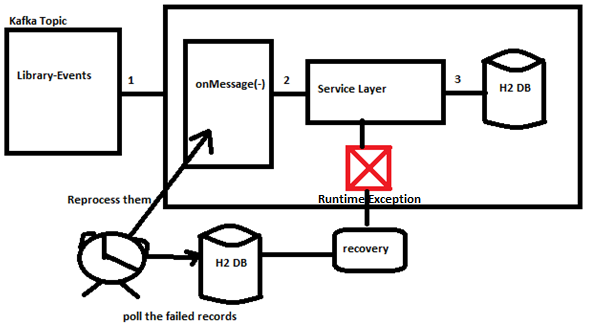
**Option1:** Publish the failed messages into a Retry topic.

**Option2:** Saved the failed message in a DB and retry with a scheduler.

**Option1:** Publish the failed messages into Retrytopic.



**Option2:** Saved the failed message in a DB and retry with a scheduler.

****

**How spring boot auto configuration works:**

Look at the **KafkaAutoConfiguration**.java file, this file will take care of the providing the necessary configuration of kafka.