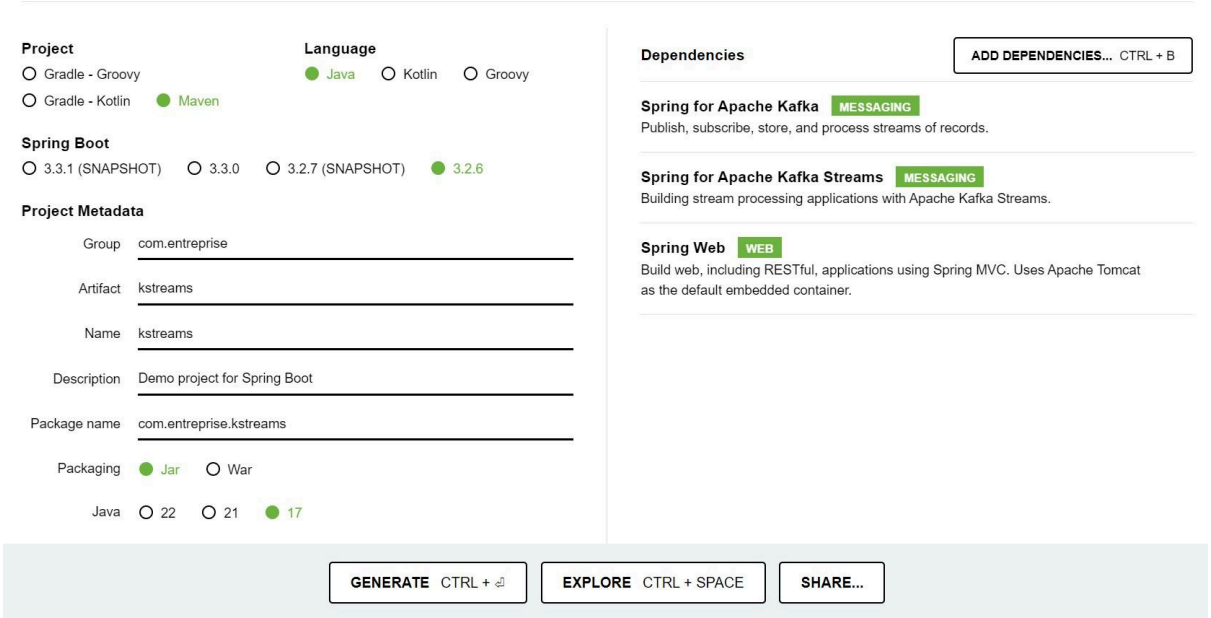
Atelier 5 : Stream API - Prise en main par la pratique.

Kafka Stream Basique.

On commence initialise un nouveau projet spring avec les dépendances suivantes :

- Spring for Apache Kafka
- Spring for Apache KAfka Streams
- Spring web



Package

```
package com.entreprise.kstreams.basiquekstreams;

import java.util.Properties;
import java.util.UUID;
import javax.print.DocFlavor.INPUT_STREAM;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.Produced;
public class Basique {
        public static final String INPUT_TOPIC="string-input";
```

```
	public static final String OUTPUT_TOPIC="string-output";

	public static void main(String[] args) {
		Properties props = new Properties();

		props.put(StreamsConfig.APPLICATION_ID_CONFIG,
UUID.randomUUID().toString());
		props.put("bootstrap.servers", "localhost:9092");


		props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,Serdes.String().getClass().get
Name());
		props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass().getName());


		// une topologie : ens des trans  faites sur les données.
		final StreamsBuilder builder = new StreamsBuilder();

		final KStream<String, String> source = builder.stream(INPUT_TOPIC);

		//Ecriture d'une topologie
		source.to(OUTPUT_TOPIC,Produced.with(Serdes.String(),Serdes.String()));
		KafkaStreams kafkastream = new KafkaStreams(builder.build(), props);
		kafkastream.start();
	}
}
```

Ensuite on lance un consumer console sur le topic string-output et un producer console sur le topic string-input et on observe la propagation des messages d'un topic à l'autre.

kafka-console-producer.bat –bootstrap-server localhost:9092 –topic string-input –property parse.key=true –property key.separator=,

On peut ensuite réaliser des transformations :

Le filtre

```
		//Transformation : Ajouter un filtre
		source = source.filter((k,v) -> {return v.length() > 5 ? true : false ; } );
```

On réalise ensuite le test en envoyant des données de valeurs inferieures et superieures à 5.
Une autre transformation parmis les plus courantes est la MapValue:

Il s’agit d’une simple transformation sur les données.

MapValues

```
//La transformation MapValue - transformation de données
                source = source.mapValues( v -> {return v.toUpperCase(); } );
```

FlatMap : À partir d’un message d’entrée avoir plusieurs records.

```
                //flatMap - prend en entrée une chaine et split la chaine
                source = source.flatMap((k,v) -> {
                        String[] tokens = v.split(" ");
                        List<KeyValue<String, String>> result = new ArrayList<>(tokens.length);
                        for (String token : tokens) {
                                result.add(new KeyValue<>(k, token));
                        }
                        return result ;
                } );
```

Transformation ForEach : Faire une action par message :

```
                //Foreach : Pour faire une action par message

                source.foreach( (k,v) ->
                        { System.out.println(v); }
                );
```