

Atelier 2 : Implémentation en Java.

Objectif : Mise en œuvre en Java des concepts de **producer** et **consumer**.

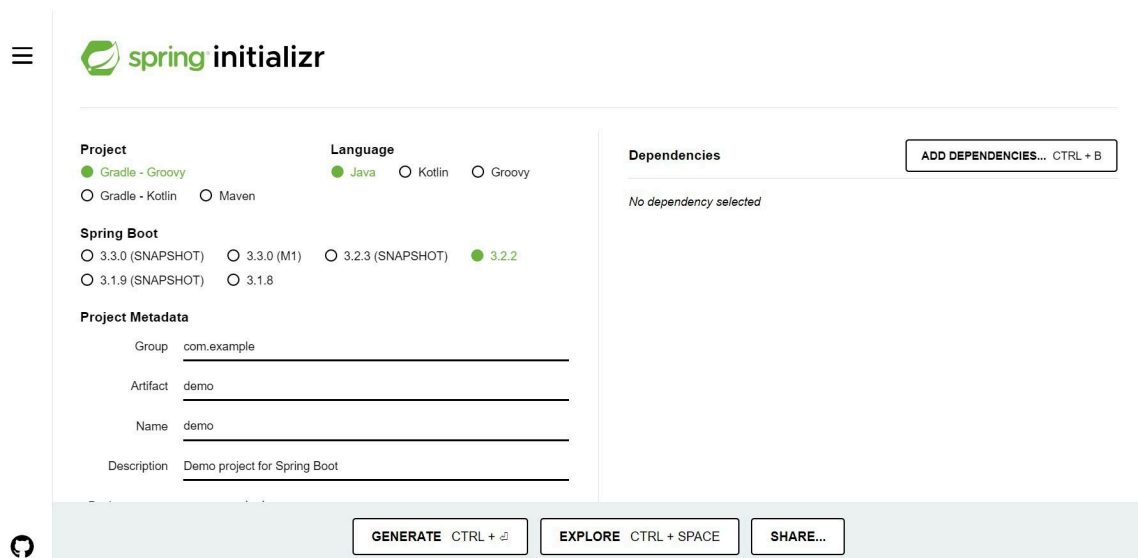
Dans cet atelier nous allons implémenter en Java (et en se basant sur le framework Spring), les concepts de base de la technologie Kafka.

Pré-requis :

Quelques pré-requis sont nécessaire pour cet atelier :

- Java
- Un IDE (Eclipse de préférence)
- Atelier 1

Étape 1 : Création du projet et ajout des dépendances.



The screenshot shows the Spring Initializr web application interface. The top navigation bar includes the Spring logo and the text 'spring initializr'. On the right, there are icons for settings and a dark mode toggle. The main content area is divided into several sections:

- Project:** Includes radio buttons for 'Gradle - Groovy' (selected), 'Gradle - Kotlin', and 'Maven'.
- Language:** Includes radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'.
- Spring Boot:** Includes radio buttons for versions '3.3.0 (SNAPSHOT)', '3.3.0 (M1)', '3.2.3 (SNAPSHOT)', '3.2.2' (selected), and '3.1.9 (SNAPSHOT)', '3.1.8'.
- Project Metadata:** Includes input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), and 'Description' (Demo project for Spring Boot).
- Dependencies:** Includes a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'.

At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

On ajoute ensuite l'ensemble des dépendances nécessaires au projet :

- Spring starter Web
- Spring kafka

Etape 2 : Le package producer

On commence par créer une classe de configuration du Producer : **KafkaProducerConfig**

```
package com.entreprise.kafkaIntegration.producer;

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;

@Configuration
public class KafkaProducerConfig {
    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();

        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

On crée une classe qui intègre une méthode sendMessage, permettant l'envoi de messages sur le broker kafka. C'est donc la classe **MessageProducer**.

```
package com.entreprise.kafkaIntegration.producer;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
public class MessageProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage(String myorsystopic, String message) {
        kafkaTemplate.send(myorsystopic, message);
    }
}
```

Etape 3 : Package Controller

Cette classe Controller représente le point d'entrée de l'application. Le méthode sendMessage définie en son sein est mobilisée à partir du path uri

http://localhost:8080/send?message=first_message_test_1

```
package com.entreprise.kafkaIntegration.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.entreprise.kafkaIntegration.producer.MessageProducer;

@RestController
public class KafkaController {

    @Autowired
    private MessageProducer messageProducer;

    @PostMapping("/send")
    public String sendMessage(@RequestParam("message") String message) {
        messageProducer.sendMessage("myorsystopic", message);
        return "Message sent :" + message;
    }
}
```

Etape 4 : Package Consumer

Classe de configuration du consumer : ConsumerConfig

```
package com.entreprise.kafkaIntegration.consumer;

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.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;

public class KafkaConsumerConfig {

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        Map<String, Object> configProps = new HashMap<>();

        configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ConsumerConfig.GROUP_ID_CONFIG, "my-group-id");
        configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

        return new DefaultKafkaConsumerFactory<>(configProps);
    }

    public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}
```

MessageConsumer intégrant la méthode **listen**, qui consomme le message envoyé par le Producer.

```
package com.entreprise.kafkaIntegration.consumer;

import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
public class MessageConsumer {

    @KafkaListener(topics = "myorsystopic", groupId = "my-group-id")
    public void listen(String message) {
        System.out.println("Received message : " + message);
    }
}
```

Exécution de la Spring-Boot Application.

On teste l'envoi de la requête sur Postman

http://localhost:8080/send?message=first_message_test_1

On consulte la console

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/send?message=first_message_test_1`. The request is configured with the following query parameters:

Key	Value	Description
message	first_message_test_1	

The response status is `200 OK` with a time of `510 ms` and a size of `198 B`. The response body is displayed in the 'Text' tab, showing the message: `Message sent :first_message_test_1`.