

Spring Kafka

Gold - Chapter 7 - Topic 1

Selamat datang di **Chapter 7 Topic 1**
online course **Back End Java** dari
Binar Academy!



Aloha, Selamat Datang di Chapter 7 ☺☺

Pada chapter sebelumnya, kita sudah kupas banyak mengenai security, Java logging, Docker, deployment sampai dengan CI/CD.

Pada chapter ini, kita bakal ngobrolin mengenai microservice dan fitur-fitur penunjangnya, seperti Spring Kafka dan Spring Gateway.

Khusus pada topic pertama, kita mengelaborasi tentang Spring Kafka dulu. Kafka ini termasuk salah satu tools yang cukup terkenal untuk pemrosesan asynchronous lho. Gimana, kamu penasaran kan? Let's go!



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Spring Kafka Introduction
- Penggunaan Spring Kafka pada use case
- Spring Kafka configuration
- Implementasi Spring Kafka



Tahukah kamu, kalo ada satu platform yang bisa nge-publish satu juta message per detik?

Yesh! Itulah si Kafka. Dengan kecanggihannya, doi bisa melakukan hal tersebut secepat kilat ✨

Daripada penasaran, yuk kita bahas di **Spring Kafka introduction~**



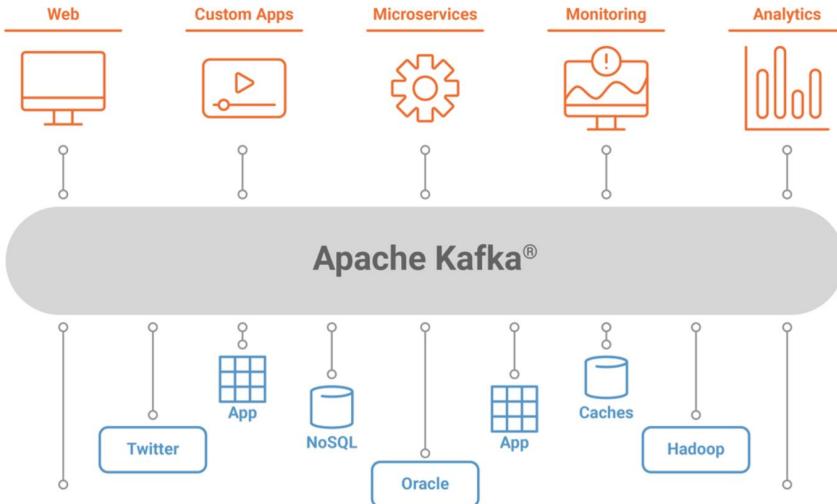
Mari kenalan dengan Kafka~

Sesuai dengan mottonya yang berbunyi “Distributed streaming platform”, Kafka adalah platform yang digunakan untuk melakukan proses publish dan subscribe stream/data.

Kayak gimana tuh? Sederhananya, publish adalah kegiatan mengirim data, sedangkan subscribe adalah kegiatan menerima data.

Kabar baiknya, Kafka bersifat open source dan free loh! Mantap banget nggak tuh□





Ada Kafka, ada database~

Digunakan untuk memanggil dan menerima data? Merasa familiar nggak? Yup, mirip seperti database! Terus bedanya sama database apa dong?

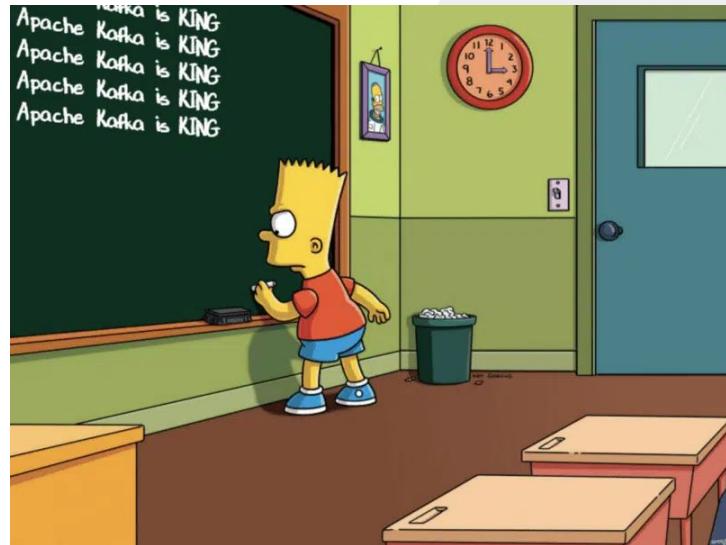
Kalo di database, setelah data tersimpan, kita perlu tarik data tersebut untuk mengambilnya.

Bedanya dengan Kafka, ketika kita mengirim data ke aplikasi (yaitu Kafka), maka data tersebut akan diterima ke semua aplikasi yang melakukan subscribe.

Udah kejawab kan penasarananya tentang konsep subscribe? □

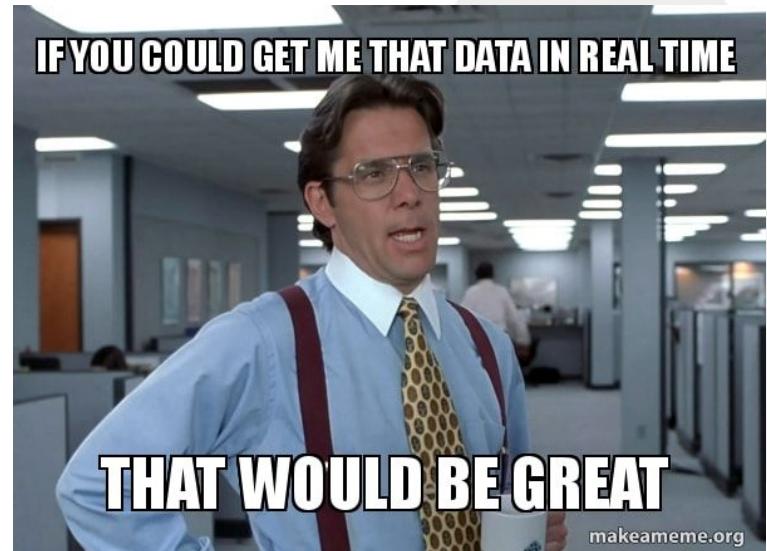
Mirip dengan konsep subscribe di Youtube, aplikasi yang melakukan subscribe tinggal duduk manis aja nunggu data~

Berbeda dengan sistem database di mana kita perlu melakukan query pada database jika kita membutuhkan data.



Lebih lanjut, Kafka dapat menerima, merekam dan mem-publish message dalam skala yang sangat besar, yakni lebih dari satu juta message per detik. Gokil kan?

Oleh karena itu, Kafka ini selain cepat, juga besar dan sangat bisa diandalkan, gengs!

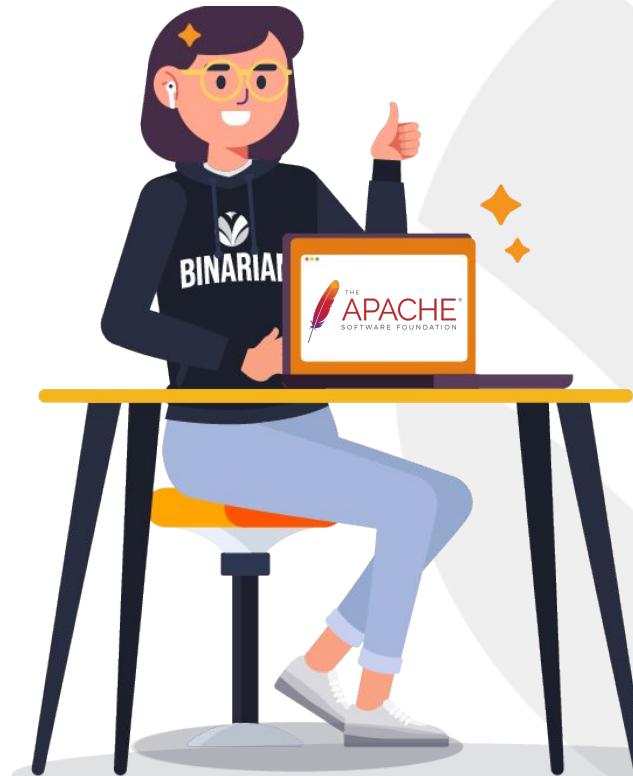


Canggih-canggih seperti ini, emang siapa pengembangnya? □

Kafka awalnya dikembangkan oleh tim developer dari LinkedIn loh, guys. Tujuannya adalah untuk melakukan data log aggregation dengan kecepatan tinggi.

Setelah dirilis, project Kafka didonasikan ke Apache Software Foundation, makanya namanya ada unsur Apache juga.

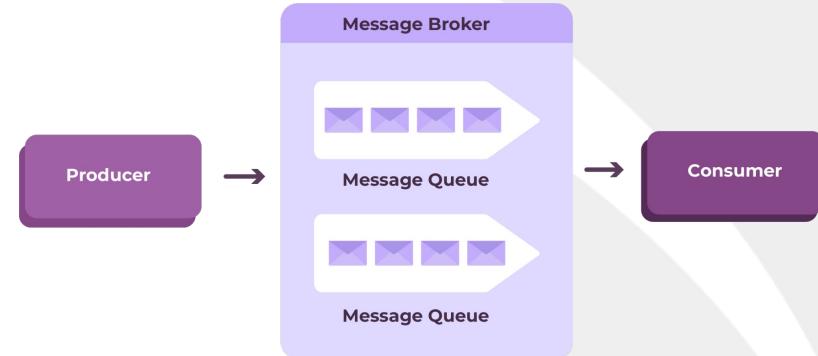
Selain itu, Kafka dibuat dengan bahasa pemrograman Scala dan Java. Karenanya, untuk menggunakan Kafka, kita perlu menginstall aplikasi Java, gengs.



Apa sih fungsi Kafka?

Kafka adalah salah satu contoh dari message broker yang lagi kekinian banget nih!

Emang apa sih message broker? Singkatnya, message broker merupakan sebuah middleware untuk menghubungkan pengirim message dan penerima message.



Ada message broker lain nggak?

Nah, selain Apache Kafka, terdapat message broker lain yaitu RabbitMQ & ActiveMQ, gengs.

Tapi tenang, nanti kita bakal bahas lebih dalam tentang message broker di Chapter 8. Sekarang kita menyelam lagi~



Mari kita intip istilah yang ada di Kafka ☐

Ada istilah yang berkaitan dengan Kafka dan perlu kamu ketahui nih, guys. Ada apa aja? Yuk, kita gali satu per satu

- **Producer**

Proses atau sistem yang dapat mem-publish data ke suatu topic.

- **Consumer**

Proses atau sistem yang dapat melakukan subscription ke satu atau lebih topic dan mengolah data-data dari topic tersebut.

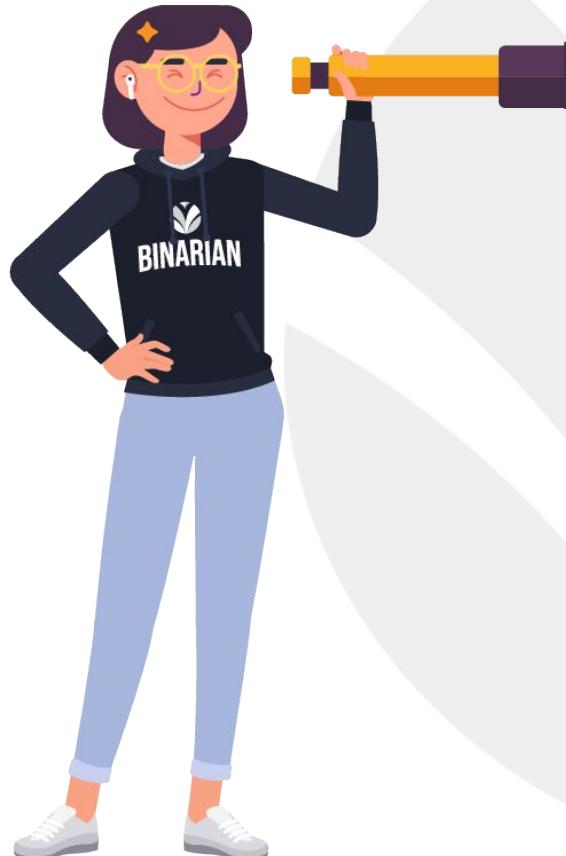
Kuterima suratmu telah
kubaca dan aku mengerti~



- **Topic**
Nama dari sebuah feed di mana pesan/data disimpan.
- **Broker**
Instance Kafka yang berjalan di satu mesin.
- **Cluster**
Kelompok dari broker-broker yang saling bekerjasama.



- **Partition**
Pengelompokan data topic yang dipecah menjadi bagian-bagian kecil. Misalnya, suatu topic menyimpan informasi user login, maka data-data pada topic dapat dibagi berdasarkan huruf awal dari username.
- **Offset**
Array index yang digunakan oleh Kafka sebagai unique identifier untuk setiap data pada satu partition.



Kenapa kita belajar Kafka? Karena ada kelebihannya!

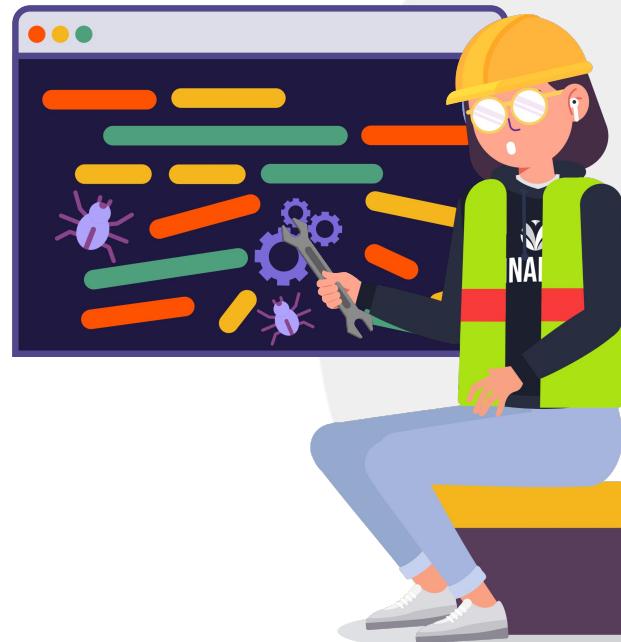
Kurang lebih ada 3 kelebihan yang dapat kita simak berikut ini, guys~

- **Distributed**
Kafka dapat menyimpan, menerima dan mengirim data atau pesan dari berbagai node.
- **Horizontally-scalable**
Kafka dapat bekerja dengan baik dalam suatu kelompok atau cluster, sehingga dengan bertambahnya volume dan kecepatan aliran data yang harus diproses, kita hanya perlu menambah mesin baru saja pada cluster tanpa harus melakukan vertical-scaling.



- **Fault-tolerant**

Jika terjadi masalah pada satu atau lebih server, maka akan di-switch/handle oleh server lain. Keseluruhan sistem tidak terganggu deh!



Bahas Kafka udah, cuss kita kaitkan ke Java-nya!

Spring untuk Apache Kafka atau bisa kita sebut Spring Kafka, dilakukan dengan menerapkan konsep core Spring (dependency injection dan declarative) untuk development message berbasis Kafka.

Spring Kafka menyediakan model pemrograman template Spring dengan KafkaTemplate dan Message-driven POJO melalui annotation @KafkaListener.



Spring Kafka membantu kita mengontrol sebagian besar pengaturan Kafka melalui konfigurasi dan menambahkan embedded Kafka untuk testing.

Hal ini bertujuan untuk menyederhanakan error handling dan transaction management, guys.

Selain itu, Spring Kafka dapat mengurangi technical/redundant code lho! Dengan begitu, kamu ga perlu ngoding panjang-panjang deh~



Oke, kita udah tahu nih Kafka itu fungsinya untuk apa. Nah, dengan banyak kelebihan yang ciamik, Kafka tuh bisa dipakai pada case apa sih?

Kali ini, kita bakal kupas tuntas penggunaan Spring Kafka pada use case. Yuk, gelar karpet dan simak materi satu ini~



Mari kita lihat kasus penggunaan Apache Kafka yang paling umum.

1. Messaging

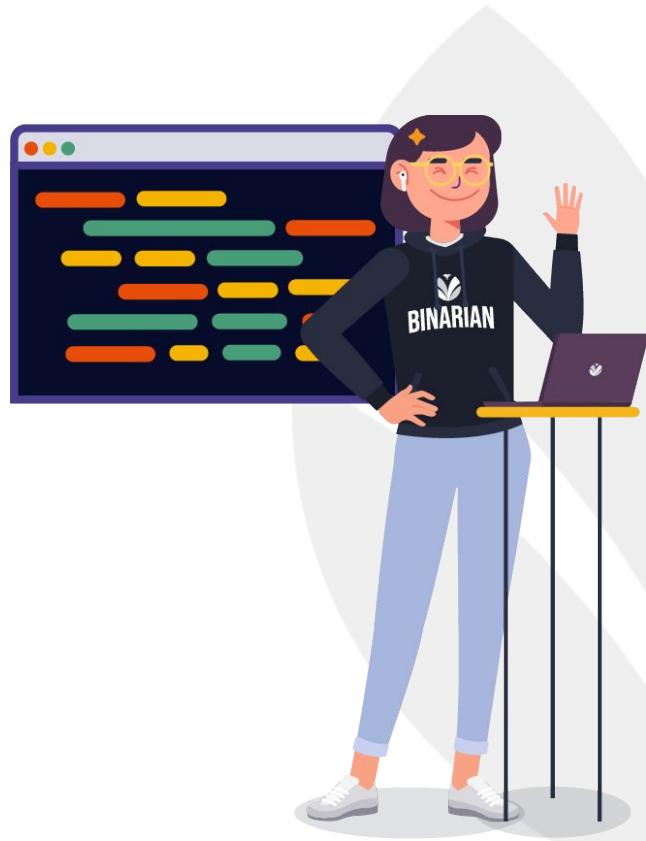
Dibandingkan dengan kebanyakan system messaging, Kafka memiliki throughput lebih baik, built-in partitioning, replication dan fault tolerant yang menjadikannya solusi yang baik untuk aplikasi pemrosesan pesan dalam skala besar.



2. Activity Tracking

Penggunaan yang paling sering dan adanya case ini adalah awal mula dibuatnya project Kafka di LinkedIn.

Berkaitan pula dengan aktivitas tracking website yang biasanya menghasilkan data dalam jumlah besar serta meng-generate berbagai message untuk setiap tampilan dan aktivitas user tertentu.



3. Data Monitoring

Kafka sering digunakan untuk data monitoring operational. Sedangkan data operational merujuk pada pemantauan berbagai hal mulai dari teknologi hingga security log.



4. Log Aggregation

Beberapa perusahaan menggunakan Kafka untuk mengumpulkan log dari berbagai service dan membuat service tersebut tersedia untuk customer mereka dalam format standar.



5. Stream Processing

Banyak user memproses data di processing pipeline yang terdiri dari beberapa tahap. Pada proses itu, raw data di-consume dari topic Kafka kemudian dikumpulkan, diperkaya, atau diubah menjadi topic baru untuk proses follow-up.



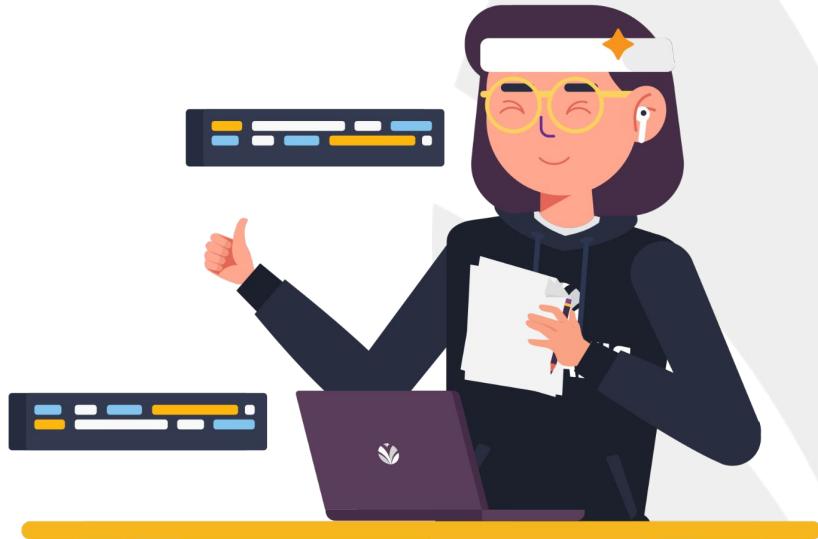
6. Event Sourcing

Kafka mendukung pengumpulan data log dalam jumlah besar. Nah, upaya menangani data log dalam jumlah besar ini menjadikannya backend yang ciamik banget untuk mem-build aplikasi.



7. Commit Log

Kafka dapat berfungsi sebagai semacam commit-log eksternal untuk distributed system. Log membantu mereplikasi data antar node dan bertindak sebagai mekanisme sinkronisasi ulang untuk node yang failed untuk me-restore datanya.



Sebelum kita hands on ke project Kafkanya, kita perlu untuk melakukan konfigurasi terlebih dahulu.

Mari kita lanjut ke **Spring Kafka configuration!**



Pertama-tama, kita perlu mendownload Kafka pada device kita.
Download pada link di bawah ini.

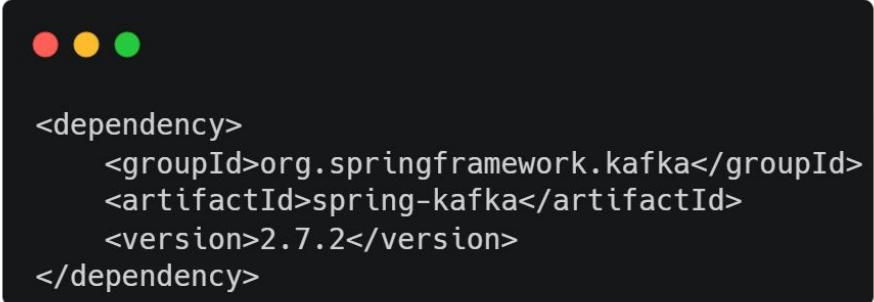
[Apache Kafka download](#)

Lalu, ikuti panduan di bawah ini untuk melakukan setting pada Kafka.

[Apache Kafka Quickstart](#)



Setelah mendownload dan menginstall Kafka, kita tambahkan dependency ke pom.xml



A decorative footer graphic consisting of three colored dots (red, yellow, green) arranged horizontally at the top left of a dark rectangular box, followed by a snippet of XML code.

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>2.7.2</version>
</dependency>
```

Sip deh! Kita mulai dengan melakukan konfigurasi untuk Topic. Cek pada gambar di bawah ini yaa!

```
● ● ●

@Configuration
public class KafkaTopicConfig {

    @Value(value = "${spring.kafka.bootstrap-servers}")
    private String bootstrapAddress;

    @Bean
    public KafkaAdmin kafkaAdmin() {
        Map<String, Object> configs = new HashMap<>();
        configs.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        return new KafkaAdmin(configs);
    }

    @Bean
    public NewTopic topic1() {
        return new NewTopic("baeldung", 1, (short) 1);
    }
}
```

Setelah itu, lakukan konfigurasi untuk Producer.

```
● ● ●

@Configuration
public class KafkaProducerConfig {

    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(
            ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            bootstrapAddress);
        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());
    }
}
```

Yang terakhir, jangan lupa untuk konfigurasi untuk Consumer~

Untuk meng-consume message, kita perlu untuk mengkonfigurasi [ConsumerFactory](#) dan [KafkaListenerContainerFactory](#).

Setelah Bean tersedia di Spring Bean Factory, POJO-based consumer bisa dikonfigurasi dengan annotation [@KafkaListener](#).

```
● ● ●

@EnableKafka
@Configuration
public class KafkaConsumerConfig {

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(
            ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
            bootstrapAddress);
        props.put(
            ConsumerConfig.GROUP_ID_CONFIG,
            groupId);
        props.put(
            ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
            StringDeserializer.class);
        props.put(
            ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
            StringDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(props);
    }

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

Fyuh! Gak kerasa udah sampai di sub topic terakhir dari topic Spring Kafka.

Gak perlu lama-lama, kita coba juga pelajari cara Implementasi Spring Kafka ya!

LET'S GO KE PART SELANJUTNYA MAS!!



Untuk membuat message, pertama kita perlu mengkonfigurasi ProducerFactory untuk menetapkan strategi pembuatan instance Kafka Producer.

Lalu, kita membutuhkan [KafkaTemplate](#) yang membungkus Producer instance dan menyediakan method yang praktis untuk mengirim message ke Kafka topic.



Producer instance ini thread safe kok guys! Jadi, menggunakan single instance di seluruh aplikasi akan memberikan performance yang lebih tinggi. Akibatnya, KafkaTemplate instance juga thread safe.



Publish message

Kita bisa mengirim message menggunakan KafkaTemplate class seperti gambar di samping.



```
@Autowired  
private KafkaTemplate<String, String> kafkaTemplate;  
  
public void sendMessage(String msg) {  
    kafkaTemplate.send(topicName, msg);  
}
```

Send API me-return object `ListenableFuture`. Kalau kita mau memblokir sending thread dan mendapatkan hasil dari message yang kita kirim, kita bisa memanggil get API dari object `ListenableFuture`.

Thread bakal nungguin hasilnya, tapiii itu bakal memperlambat producernya 😊

Perlu diingat, Kafka itu fast-stream processing platform yaa, guys. Oleh karena itu, lebih baik hasilnya ditangani secara asynchronous aja supaya message selanjutnya nggak perlu menunggu hasil dari message sebelumnya.

```
public void sendMessage(String message) {  
  
    ListenableFuture<SendResult<String, String>> future =  
        kafkaTemplate.send(topicName, message);  
  
    future.addCallback(new ListenableFutureCallback<SendResult<String, Str  
  
        @Override  
        public void onSuccess(SendResult<String, String> result) {  
            System.out.println("Sent message=[ " + message +  
                " ] with offset=[ " + result.getRecordMetadata().offset() + " ]");  
        }  
        @Override  
        public void onFailure(Throwable ex) {  
            System.out.println("Unable to send message=[ " +  
                message + " ] due to : " + ex.getMessage());  
        }  
    );  
}
```

Consuming Message

Untuk melakukan consuming message, cek gambar di samping ya bestie!

```
@KafkaListener(topics = "topicName", groupId = "foo")
public void listenGroupFoo(String message) {
    System.out.println("Received Message in group foo: " + message);
}
```

Nah, kita juga bisa mengimplementasikan multiple listeners untuk satu topic, masing-masing dengan ID grup yang berbeda. Selanjutnya, satu consumer dapat me-listen pesan dari berbagai topic.



```
@KafkaListener(topics = "topic1, topic2", groupId = "foo")
```

Spring juga mendukung pengambilan satu atau lebih header message menggunakan annotation `@Header` di listener.

```
● ● ●  
@KafkaListener(topics = "topicName")  
public void listenWithHeaders(  
    @Payload String message,  
    @Header(KafkaHeaders.RECEIVED_PARTITION_ID) int partition) {  
    System.out.println(  
        "Received Message: " + message  
        + "from partition: " + partition);  
}
```

Perlu diingat ya guys, Spring Kafka lebih baik digunakan untuk menangani volume data yang besar, data yang butuh diproses secara paralel dan sistem yang cukup kompleks karena melibatkan banyak proses dan monitoring.

Jika kamu hanya butuh pemrosesan data asynchronous saja, kamu bisa menggunakan traditional messaging lho!



Yuhuuu! Kamu udah berhasil menamatkan
Chapter 7 Topic 1 ☐

Selanjutnya, kita bakal bahas tentang Spring
Gateway.

Penasaran kayak gimana? Cuss, langsung ke topik
selanjutnya~

