



Introduction à l'événementiel

Spring Batch & Spring Cloud
Stream et Kafka

Général

- Initiation à l'événementiel
- Écosystème du cours / projet
 - Spring Boot
 - Spring Data
 - Spring Batch
 - Spring Cloud
 - Kafka

Écosystème



Spring
Boot

+



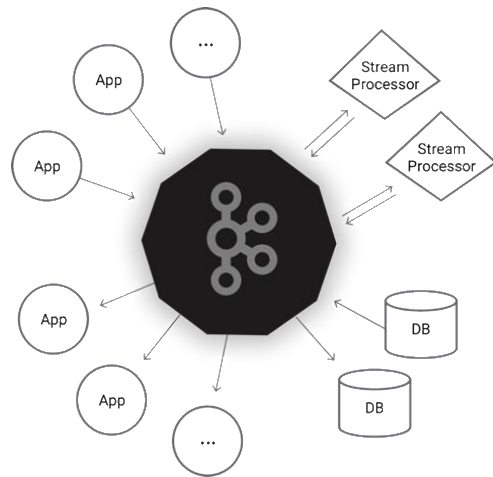
Spring
Cloud
Task /
Stream

+

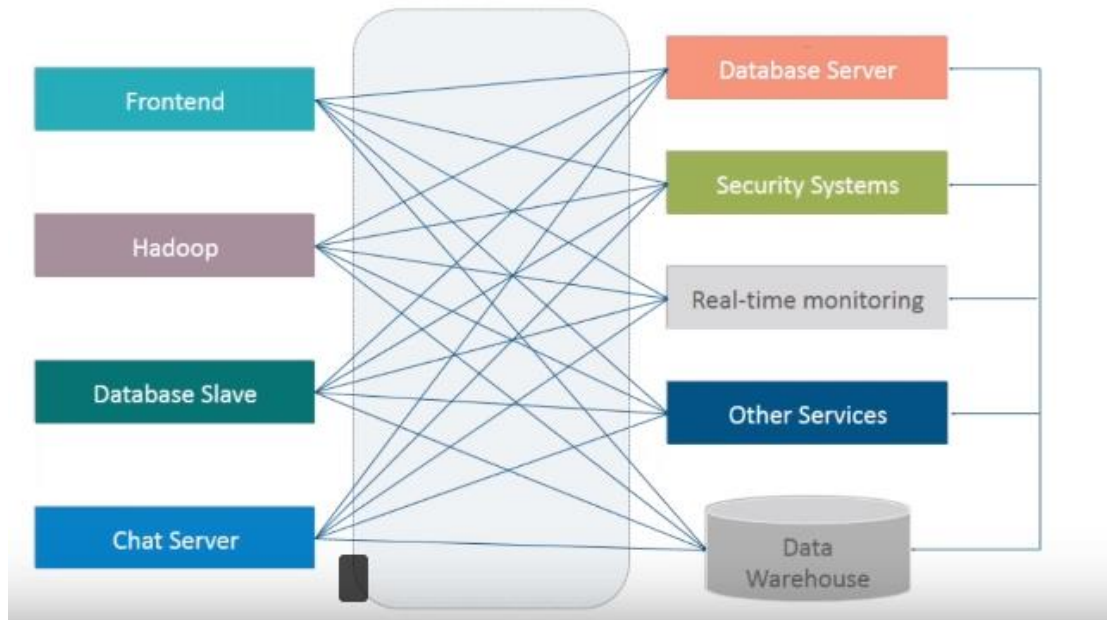


Kafka

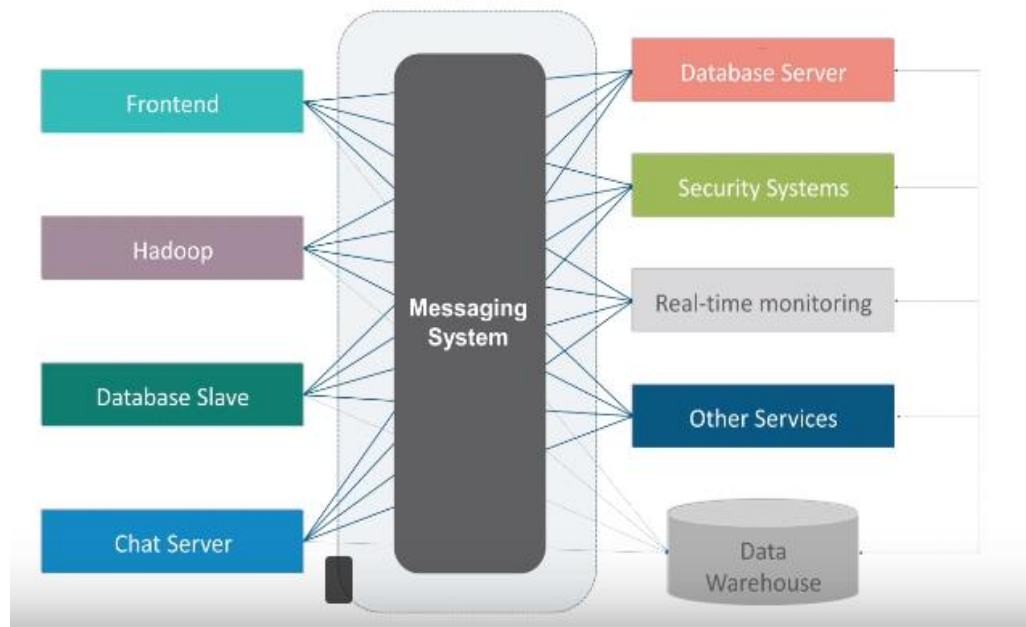
- Système de messagerie
 - Publier et s'abonner
- Distribué
- Tolérance de panne
- Évolutif (grands volumes de données)
- Temps réel
- Faible latence



Kafka



Kafka

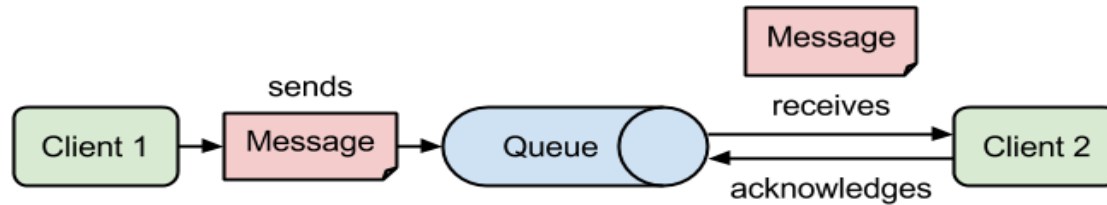


Kafka - Messaging System

1. Point to Point System
2. Publish-Subscribe System

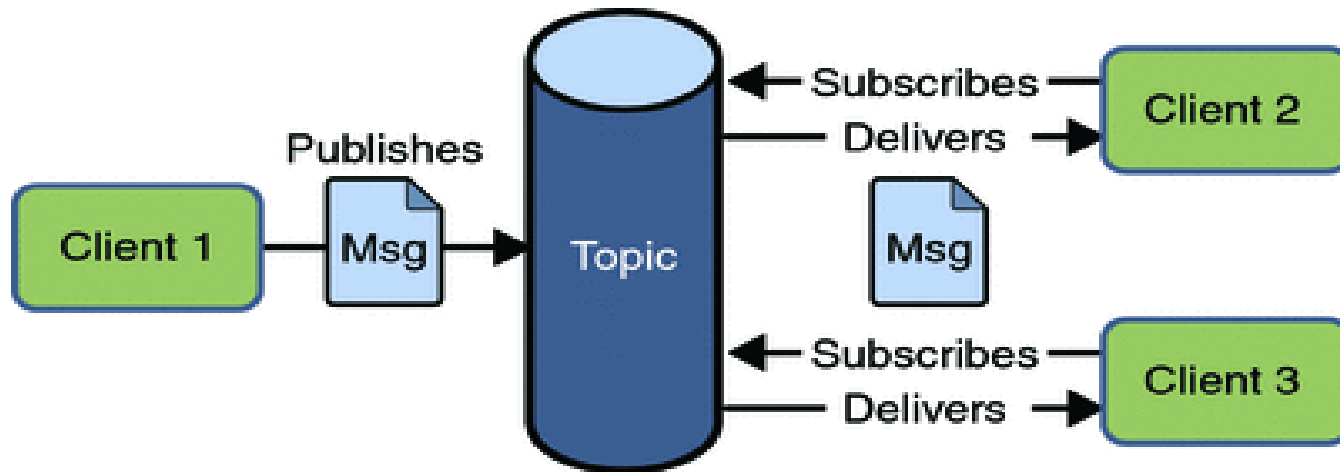
Kafka - Point to Point System

- Message dans les files d'attente
- Consommation de message par un seul consommateur

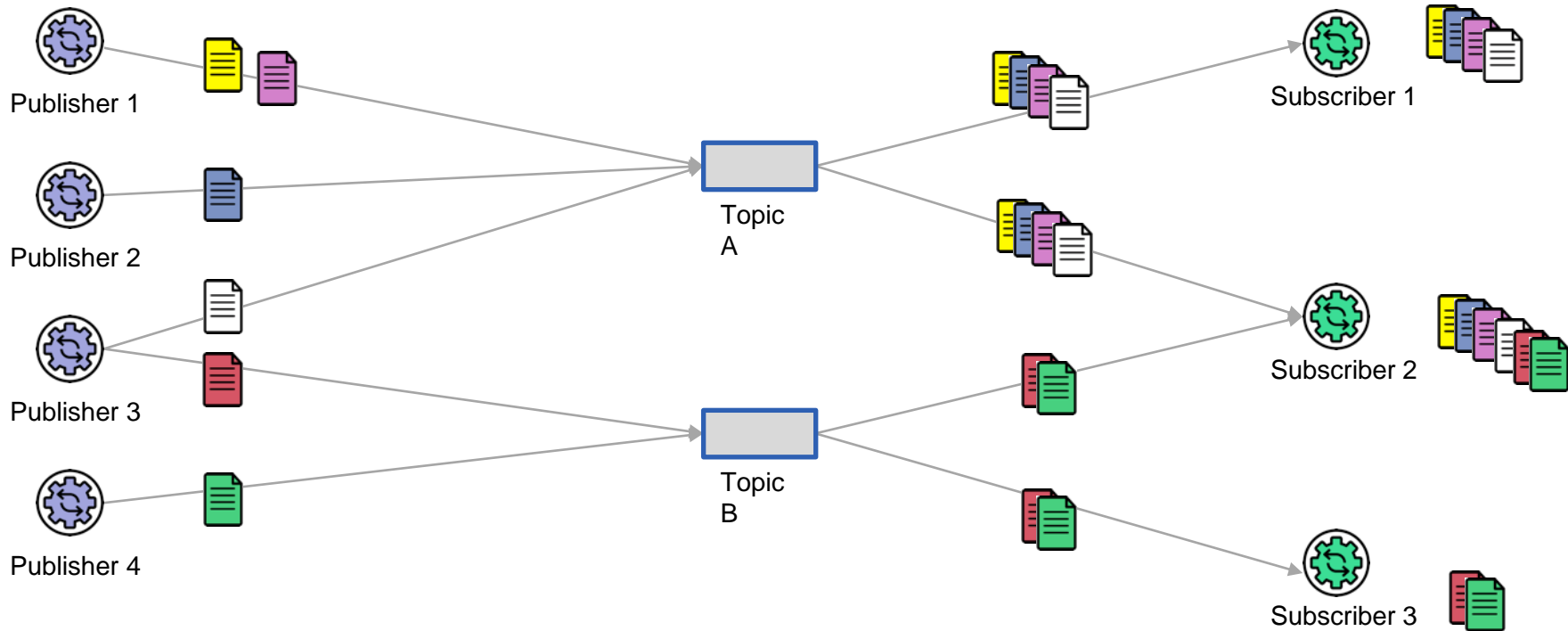


Kafka - Publish-Subscribe System

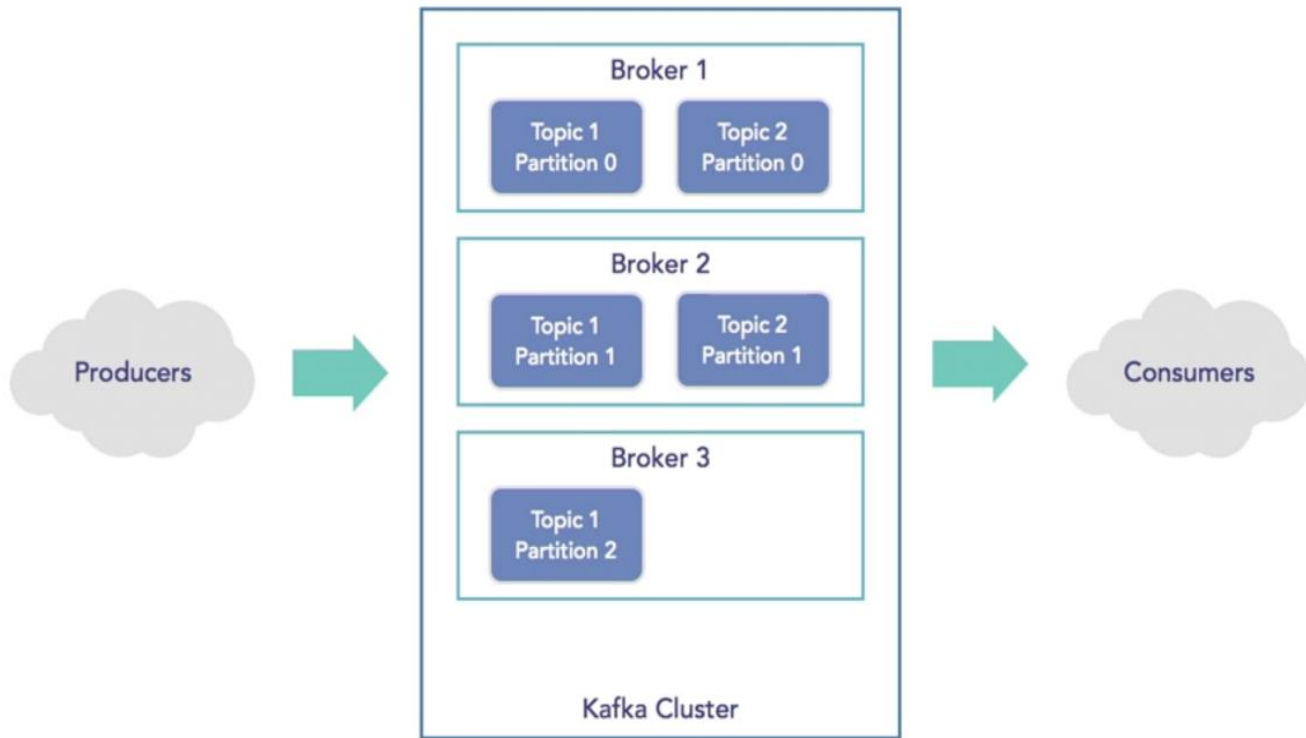
- Message dans les Topic
- Un consommateur peuvent s'abonner sur un ou plusieurs Topic



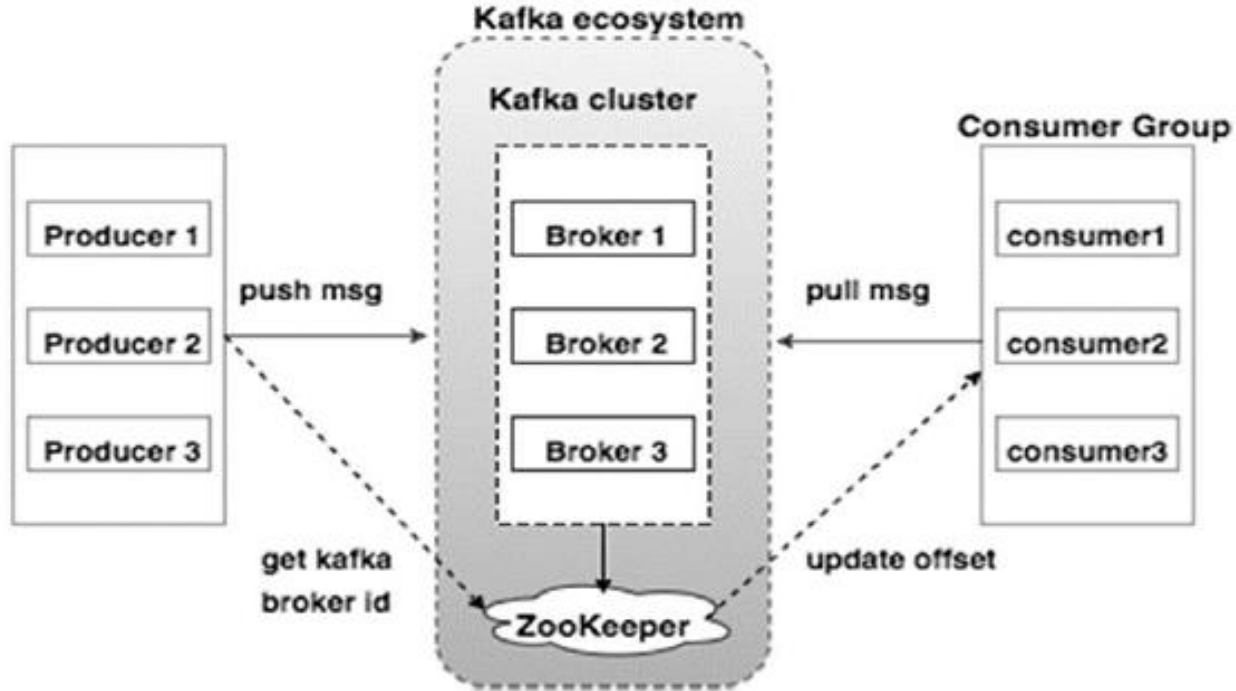
Kafka



Apache Kafka



Apache Kafka



Kafka Components

1. Topic
2. Kafka Producer
3. Kafka Consumer
4. Kafka Broker
5. Kafka Zookeeper

Implementation

<artifactId>spring-kafka</artifactId>

```
@Service
public class KafkaSender {
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;
    String kafkaTopic = "dev-topic";

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

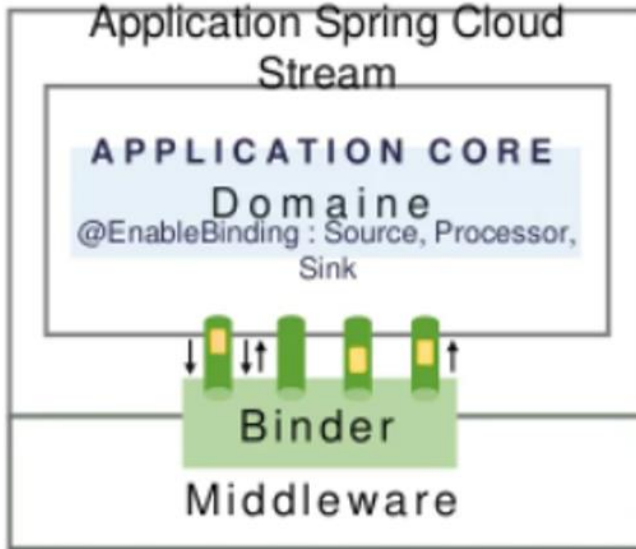
Implementation

```
@RestController
@RequestMapping(value = "/dev-kafka/")
public class ApacheKafkaWebController {

    @Autowired
    KafkaSender kafkaSender;

    @GetMapping(value = "/producer")
    public String producer(@RequestParam("message") String message) {
        kafkaSender.send(message);
        return "Message sent to the Kafka Topic developervisits-topic Successfully";
    }
}
```

Écosystème



Kafka Binder

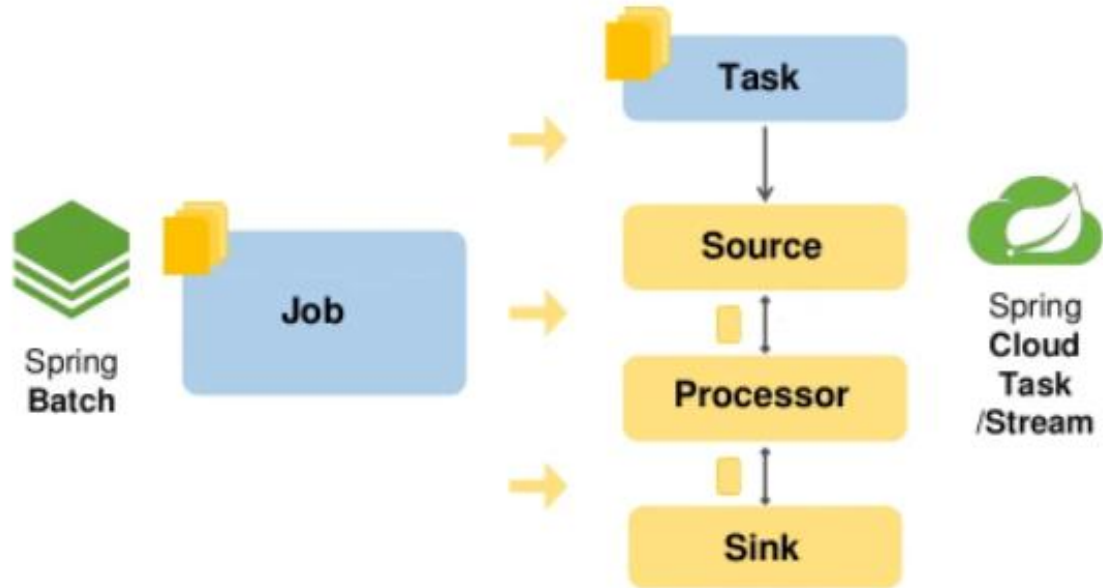


Topic

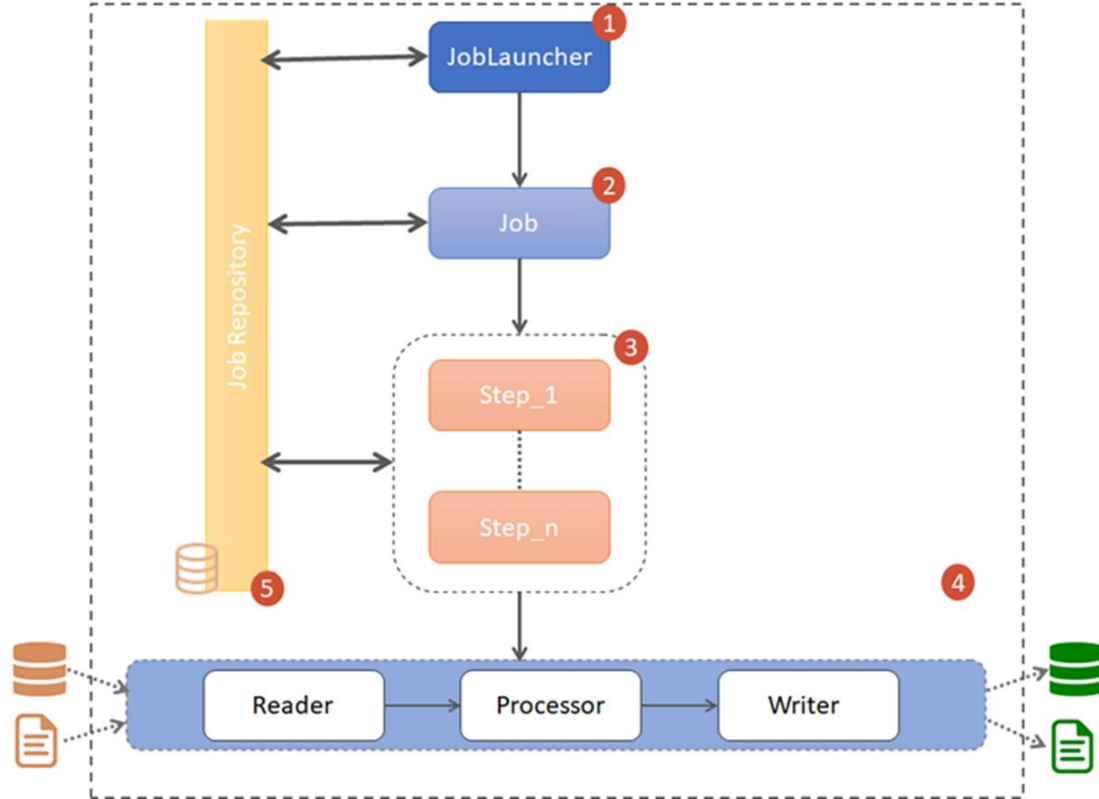
Topic

Topic

Processus

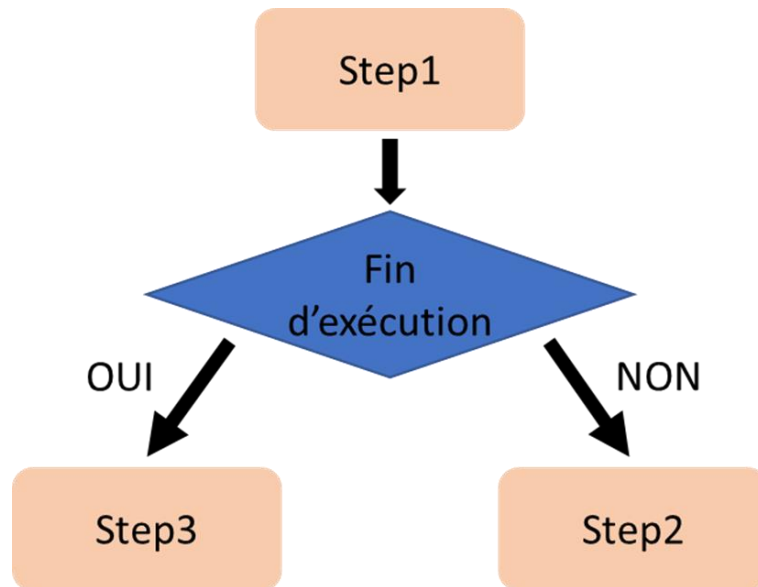


Spring Batch - Architecture



Job - Définition / Création

```
@Bean
public Job jobExample() {
    return _jobBuilder.get(" jobExample")
        .start(step1())
        .on("*").to(step3())
        .from(step1()).on("FAILED").to(step2())
        .end()
        .build();
}
```



Création d'un Step

@Bean

```
public Step step1() {  
    return _jobBuilder.get("step1")  
        .reader(new ReaderExample())  
        .processor(new ProcessorExample())  
        .writer(new WriterExample())  
        .build();  
}
```

@Bean

```
public Step step2() {  
    return _jobBuilder.get("step2")  
        .tasklet( new TaskletExample())  
        .build();  
}
```

Création d'un Step - Item<>

```
public interface ItemReader<T> {  
    T read() throws Exception, UnexpectedInputException, ParseException, NonTransientResourceException;  
}
```

```
public interface ItemProcessor<T, O> {  
    O process(T obj) throws Exception;  
}
```

```
public interface ItemWriter<T> {  
    void write(List<? Extends T> items) throws Exception;  
}
```

Création d'un Step - Tasklet

```
public Step TaskletExample implements Tasklet {  
    @Override  
    public RepeatStatus execute(final StepContribution stepContribution, final ChunkContext chunkContext) {  
        return RepeatStatus.FINISHED;  
    }  
}
```

Création d'un Step – Item*

@Bean

```
public Step step2(){  
    return this.stepBuilderFactory  
        .get("step2")  
        .<List<String>, List<Letter>>chunk(1)  
        .reader(new SimpleItemReader())  
        .processor(new SimpleItemProcessor())  
        .writer(new SimpleItemWriter())  
        .build();  
}
```

Création d'un Step – ItemReader

```
public class SimpleItemReader implements ItemReader<List<String>> {  
  
    @Override  
    public List<String> read() {  
        ArrayList<String> messages = new ArrayList<>();  
        /* ... */  
        return messages;  
    }  
}
```


Création d'un Step – ItemProcessor

```
public class SimpleItemProcessor implements ItemProcessor<List<String>, List<Letter>> {  
  
    @Override  
    public List<Letter> process(List<String> messages) throws Exception {  
        ArrayList<Letter> letters = new ArrayList<>();  
        /* ... */  
        return letters;  
    }  
}
```

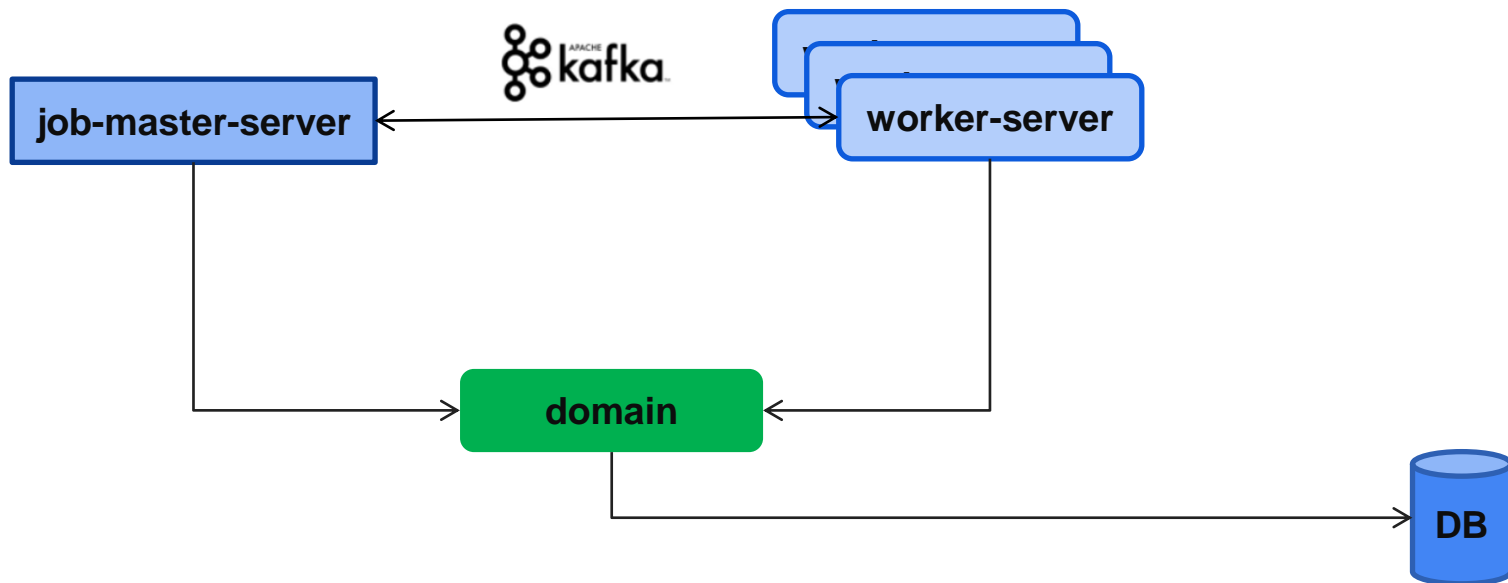
Création d'un Step – ItemWriter

```
public class SimpleItemWriter implements ItemWriter<List<Letter>> {  
  
    @Override  
    public void write(List<? extends List<Letter>> letters) throws Exception {  
        // log message  
        // save letter list in database */  
    }  
}
```

Send message kafka

- `@EnableBinding({Source.class / Processor.class / Sink.class })`
 - `@StreamListener(target = [Channel Name], condition)`
 - `@ServiceActivator(inputChannel = [Channel Name])`
-
- Remote Chunking
 - Remote Partitioning

Architecture du projet



Architecture du projet

- domain
 - Configuration de la base
 - Objets métiers
- job-master-server : permet de gérer le lancement des Jobs
 - Définition de création d'un Job
 - JobExecutionSink qui permet d'écouter des messages du « worker-server »
- worker-server : permet d'exécuter les Jobs
 - JobProcessor qui permet d'écouter / d'envoyer des messages au « job-master »

Fonctionnement

- Lancement du serveur **Kafka**
- Lancement du serveur **domain**
- Lancement du serveur **server-queue**
- Lancement de n serveurs **worker-service**

Spring Boot - Spring Cloud- Kafka
