# Introduction à l'événementiel

Spring Batch & Spring Cloud Stream et Kafka

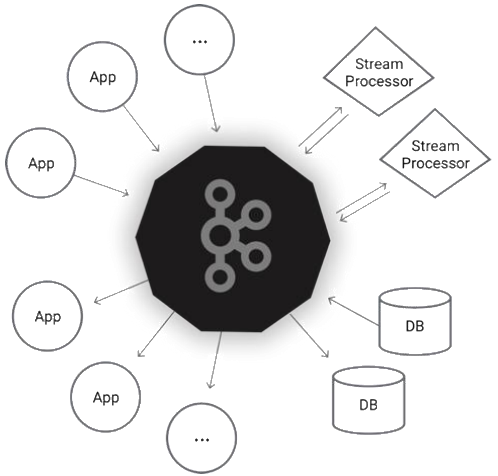
# Général

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

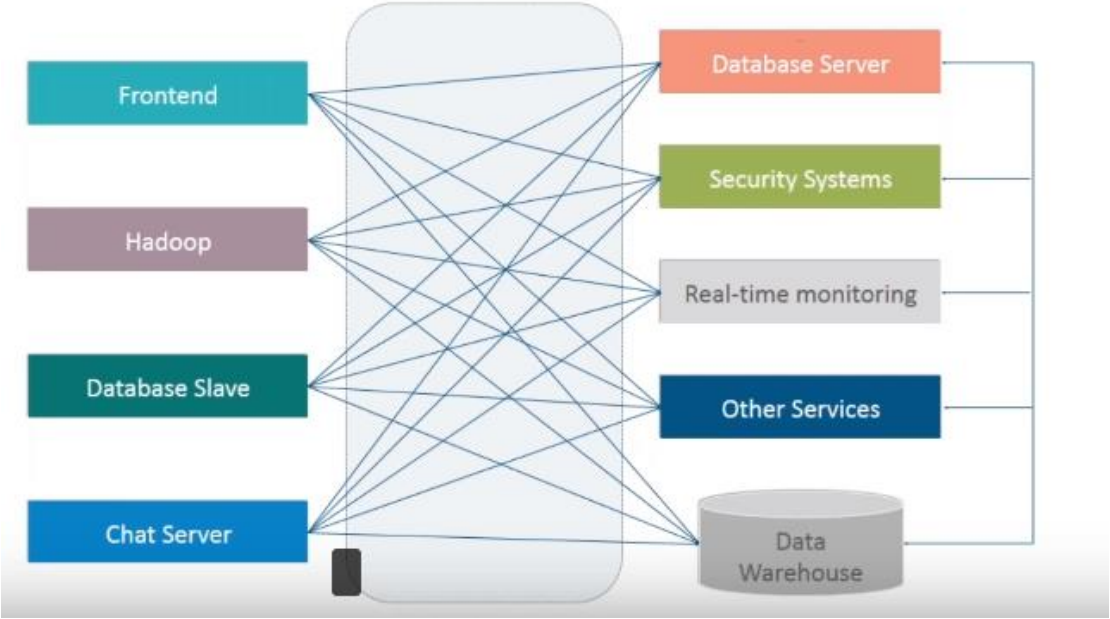
# Écosystème

# Kafka

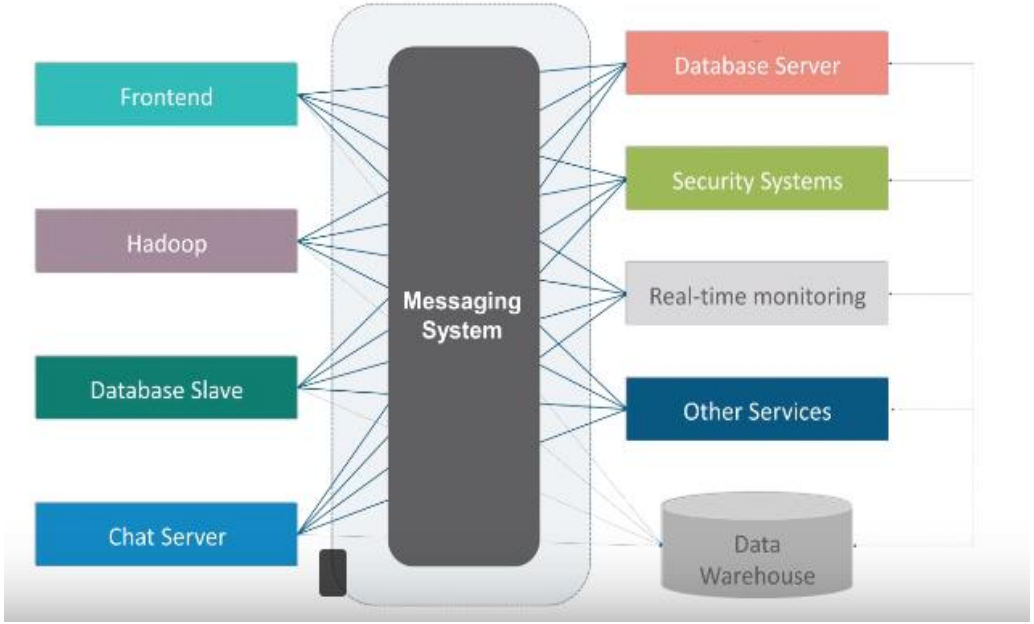
- Système de messagerie
  - Publier et s'abonner
- Distribué
- Tolérance de panne
- Évolutif (grands volumes de données)
- Temps réel
- Faible latence



# Kafka


Frontend
Hadoop
Database Slave
Chat Server
Database Server
Security Systems
Real-time monitoring
Other Services
Data Warehouse

# Kafka

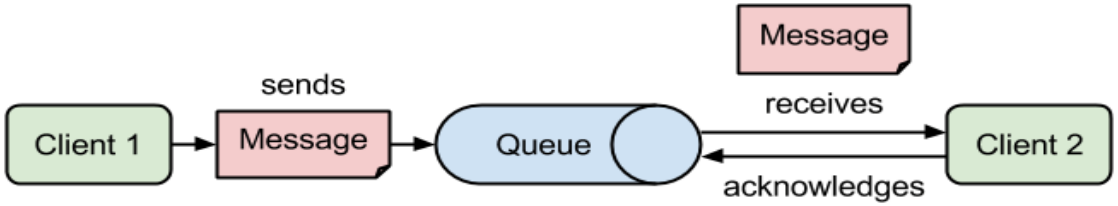

Frontend
Hadoop
Database Slave
Chat Server
Messaging System
Database Server
Security Systems
Real-time monitoring
Other Services
Data Warehouse

# Kafka - Messaging System

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

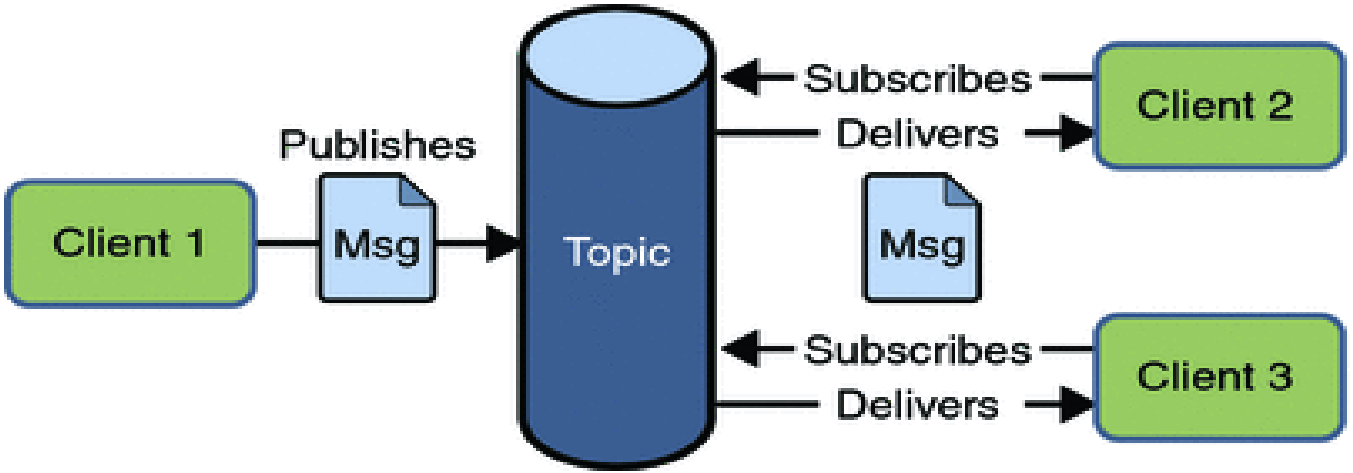
# Kafka - Point to Point System

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

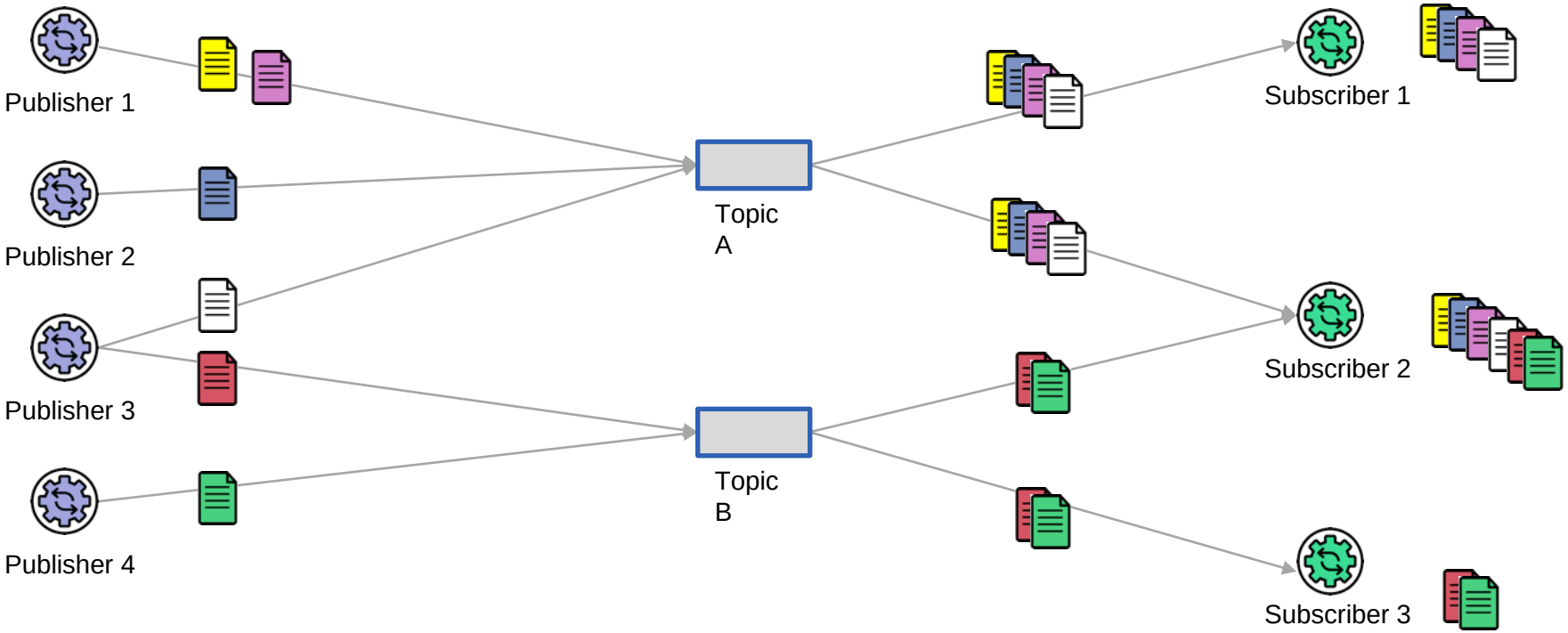


# Kafka - Publish-Subscribible System

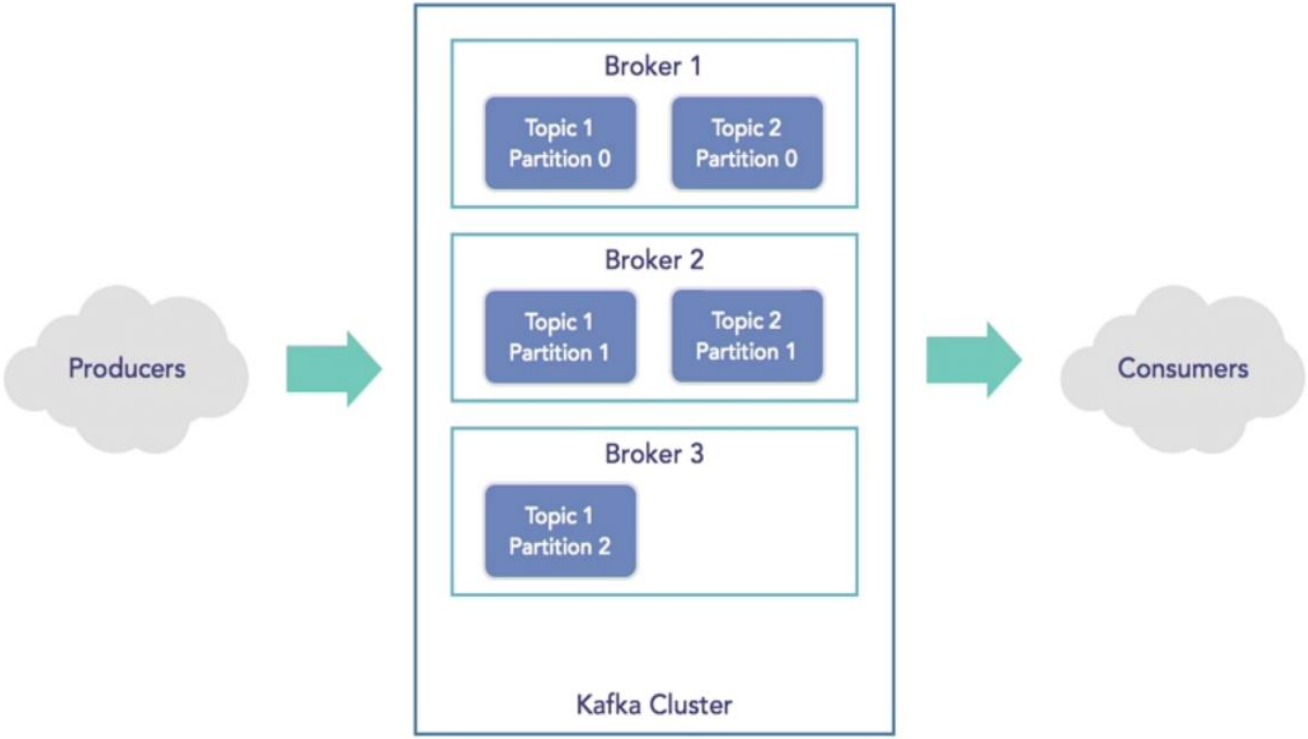
- Message dans les Topic
- Un consommateur peuvent s’abonner sur un ou plusieurs Topic



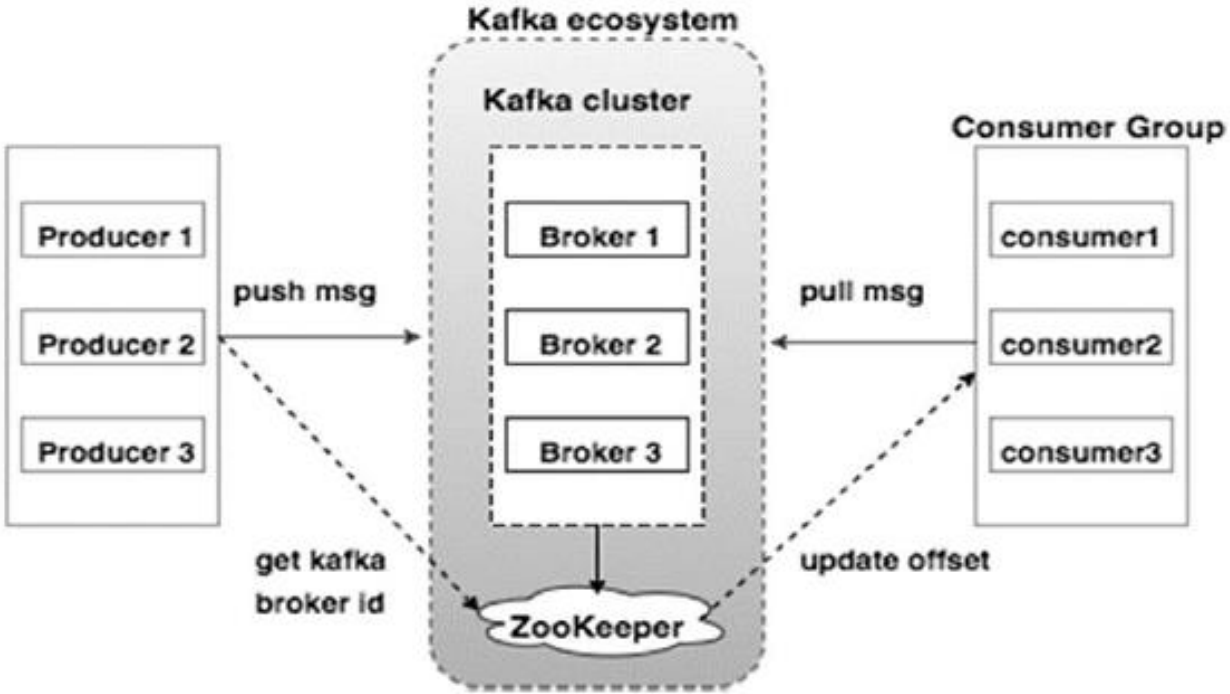
# Kafka


Publisher 1
Publisher 2
Publisher 3
Publisher 4
Topic
A
Topic
B
Subscriber 1
Subscriber 2
Subscriber 3

# Apache Kafka



# Apache Kafka



# Kafka Components

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

# Implementation

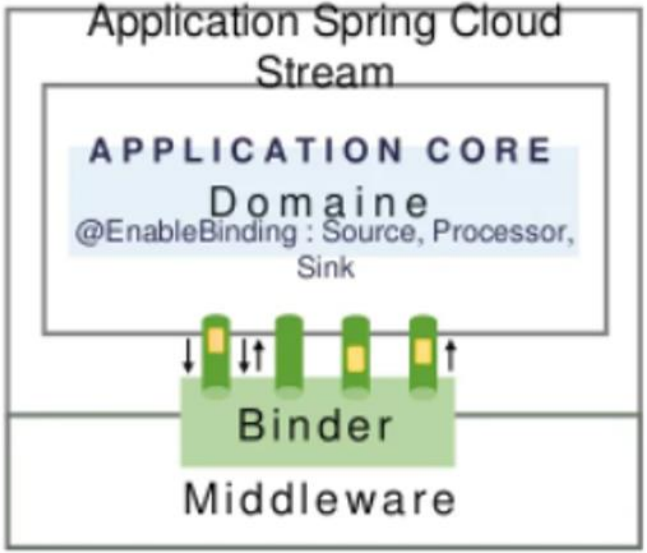
<artifactId>spring-kafka</artifactId>

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

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

# Implementation

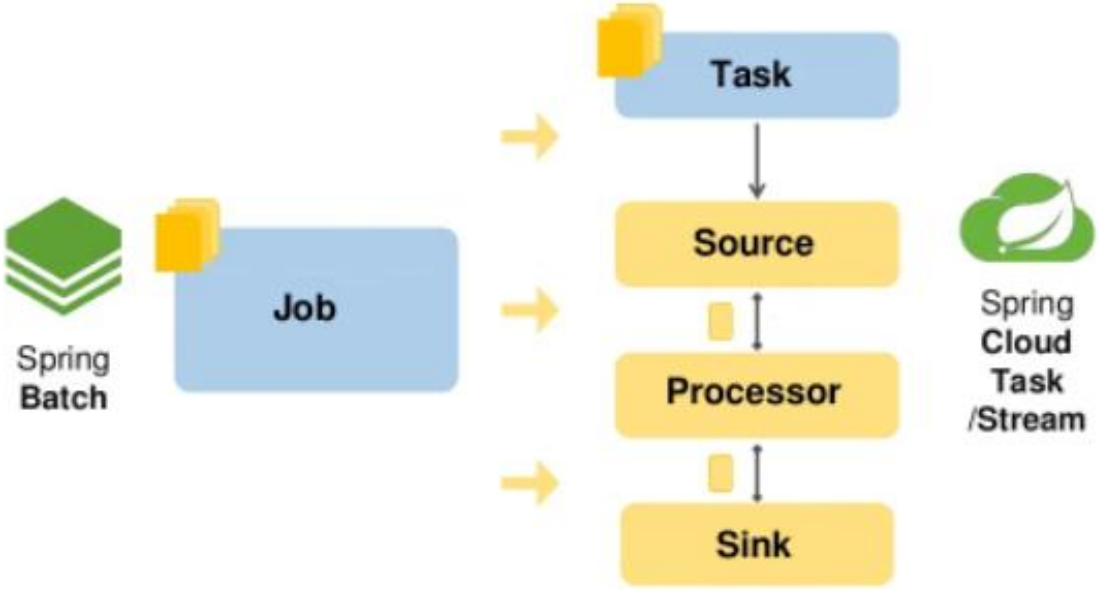
```
@RestController
@RequestMapping(value = "/dev-kafka/")
public class ApacheKafkaWebController {

    @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

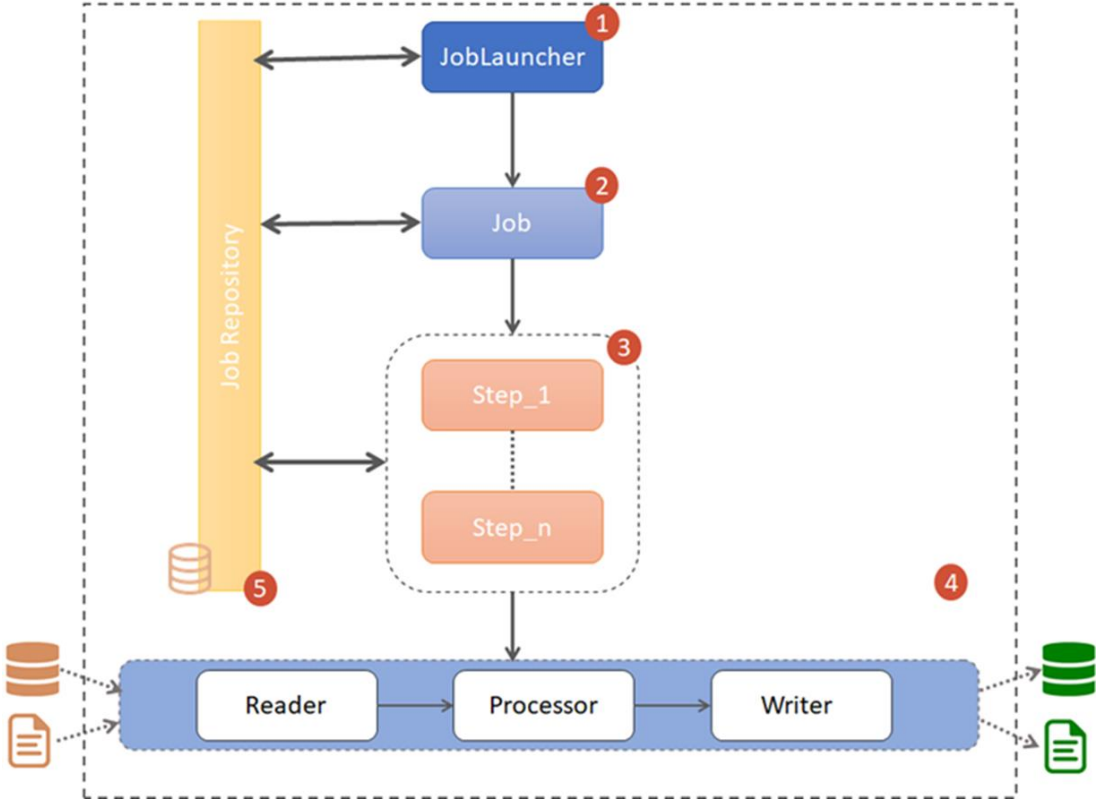



**Kafka Binder**

# Processus

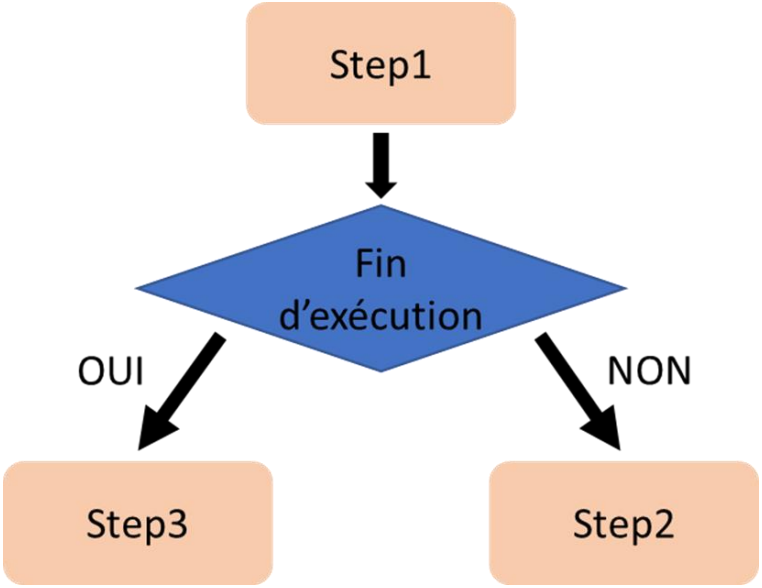

Task
Source
Processor
Sink
Job
Spring
Batch
Spring
Cloud
Task
/Stream

# 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;
    }
}
```

# ItemReader from file

```
@Bean
public FlatFileItemReader<String> reader() {
    FlatFileItemReader<String> reader = new FlatFileItemReader<>();
    /* … */
    return reader;
}
```

# 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 })
  - bind.output().send(*[Message]*)

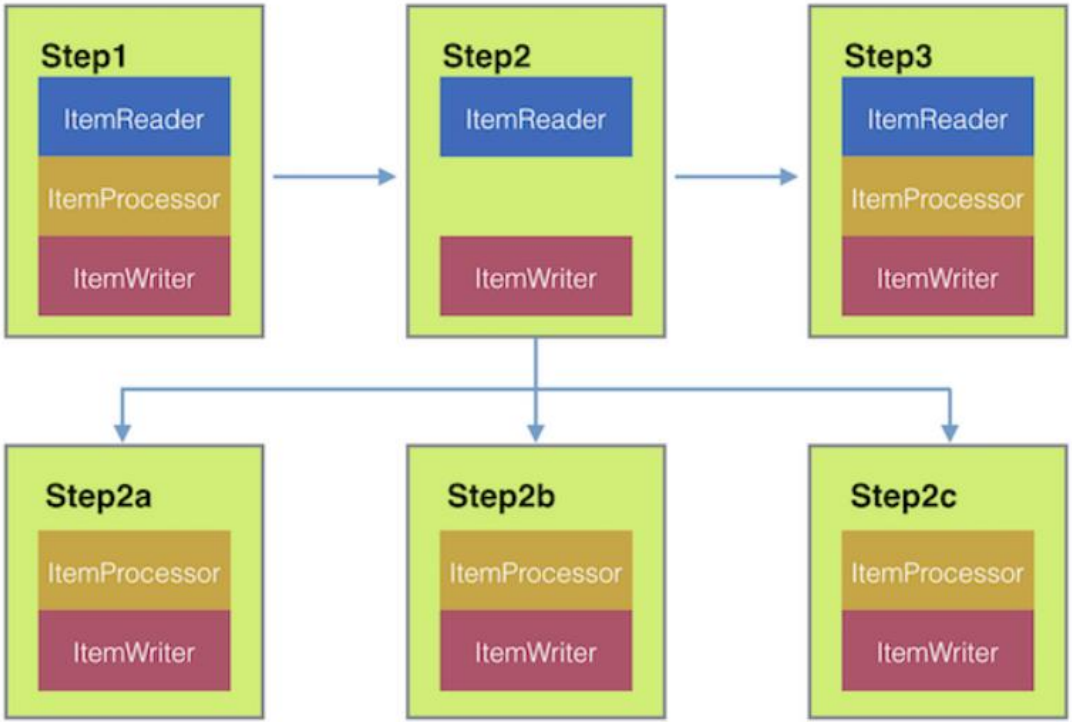
- @StreamListener(targOet = *[Channel Name]*, condition)
- @ServiceActivator(inputChannel = *[Channel Name]*)

# Send message kafka

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

- Remote Chunking
- Remote Partitioning

# Remote Chunking



# Remote Chunking - Master

```
@Bean
public Job chunkJob() {
    return jobBuilderFactory
            .get("chunkJob")
            .start(managerStep())
            .build();
}
```

```
@Autowired
private RemoteChunkingManagerStepBuilderFactory
managerStepBuilderFactory;

@Bean
public Step managerStep() {
    return this.managerStepBuilderFactory.get("managerStep")
            .chunk(10)
            .reader(new messageItemReader())
            .outputChannel(requests())
            .inputChannel(replies())
            .build();
}
```

# Remote Chunking - Master

```
@Autowired
private ConsumerFactory kafkaFactory;

@Bean
public DirectChannel requests(){ return new DirectChannel(); }

@Bean
public IntegrationFlow inboundFlow () {
    final ContainerProperties containerProps = new ContainerProperties(TOPIC);
    containerProps.setGroupId(MessageJobMasterConfiguration.GROUP_ID);

    final KafkaMessageListenerContainer container = new KafkaMessageListenerContainer(kafkaFactory, containerProps);
    final KafkaMessageDrivenChannelAdapter kafkaMessageChannel = new KafkaMessageDrivenChannelAdapter(container);

    return IntegrationFlows.from(kafkaMessageChannel).channel(requests()).get();
}
```

# Remote Chunking - Master

```
@Bean
public QueueChannel replies() { return new QueueChannel(); }

@Bean
public IntegrationFlow outboundFlow() {
    final KafkaProducerMessageHandler kafkaMessageHandler = new KafkaProducerMessageHandler(kafkaTemplate);
    kafkaMessageHandler.setTopicExpression(new LiteralExpression(TOPIC));
    return IntegrationFlows
            .from(replies())
            .handle(kafkaMessageHandler)
            .get();
}
```

# Remote Chunking - Worker

```
@Bean
public IntegrationFlow workerFlow() {
    return this.workerBuilder
            .itemProcessor(new messageItemProcessor())
            .itemWriter(new messageItemWriter())
            .inputChannel(requests()) // requests received from the manager
            .outputChannel(replies()) // replies sent to the manager
            .build();
}
```

# Remote Chunking - Worker

```
@Bean
public DirectChannel requests(){ return new DirectChannel(); }

@Bean
public IntegrationFlow inboundFlow () {
    final ContainerProperties containerProps = new ContainerProperties([TOPIC]);
    containerProps.setGroupId(MessageWorkerConfig.GROUP_ID);

    final KafkaMessageListenerContainer container = new KafkaMessageListenerContainer(kafkaFactory, containerProps);
    final KafkaMessageDrivenChannelAdapter kafkaMessageChannel = new KafkaMessageDrivenChannelAdapter(container);

    return IntegrationFlows
            .from(kafkaMessageChannel)
            .channel(requests())
            .get();
}
```

# Remote Chunking - Worker

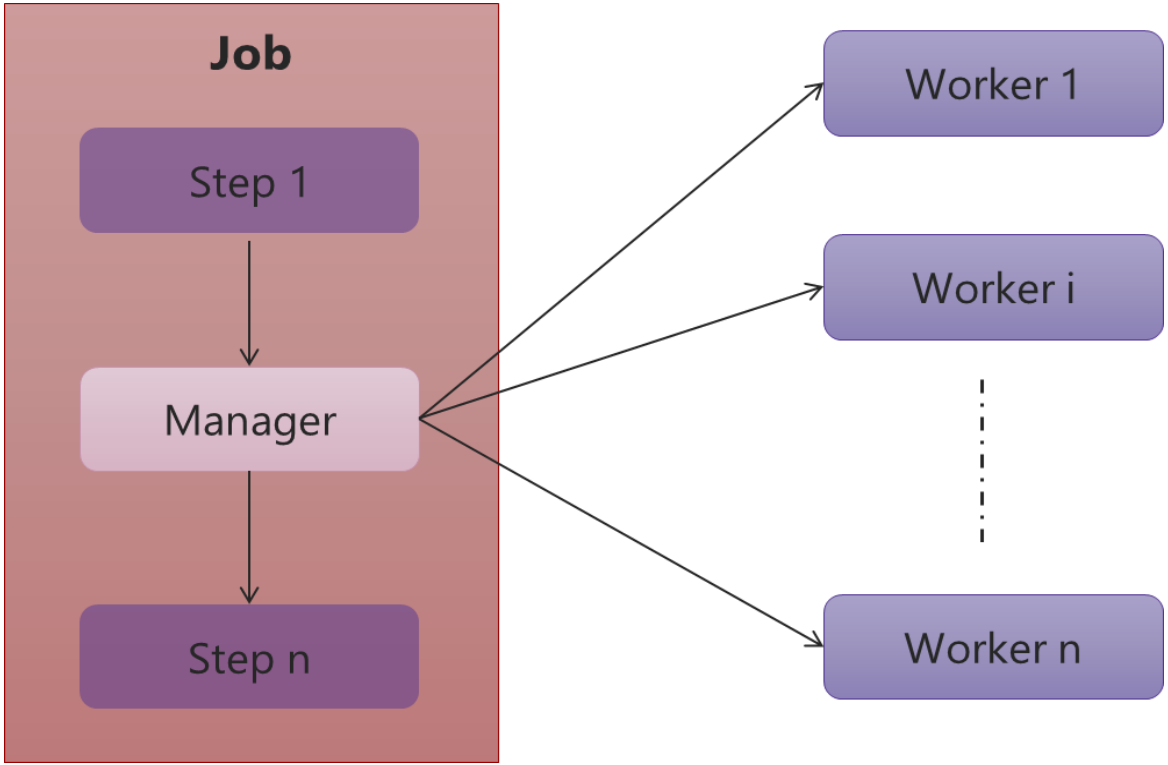
```
@Bean
public QueueChannel replies() { return new QueueChannel(); }

@Bean
public IntegrationFlow outboundFlow() {
    final KafkaProducerMessageHandler kafkaMessageHandler = new
KafkaProducerMessageHandler(kafkaTemplate);
    kafkaMessageHandler.setTopicExpression(new LiteralExpression([TOPIC]));
    return IntegrationFlows
            .from(replies())
            .handle(kafkaMessageHandler)
            .get();
}
```

# Chunking

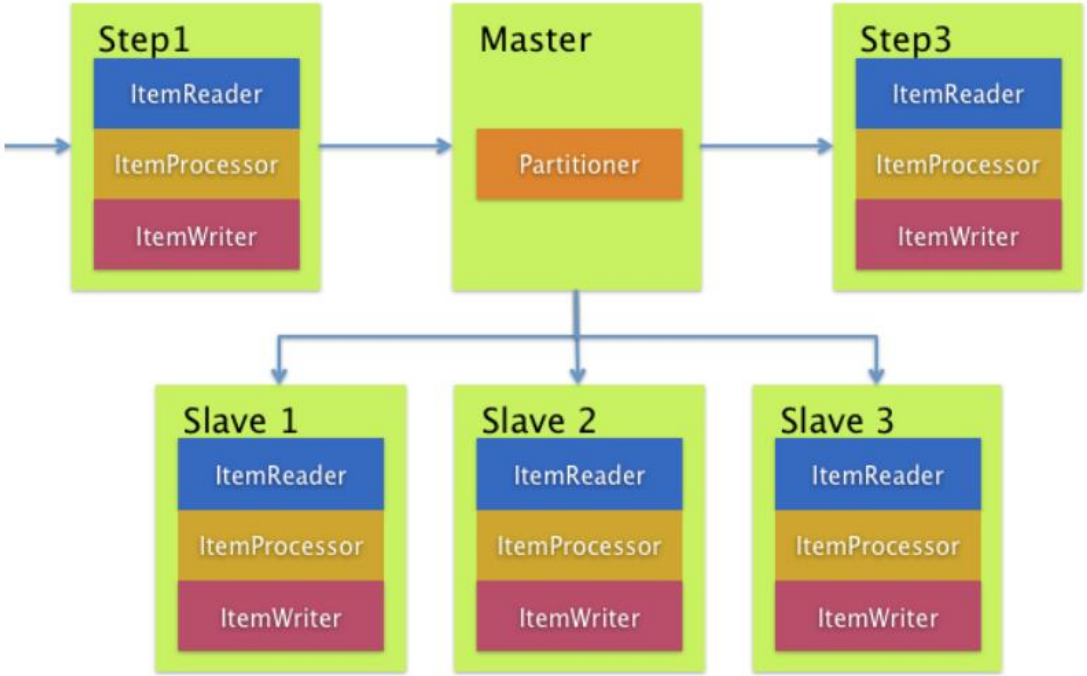
- Annotation de configuration sur la classe :
  @EnableBatchIntegration
  @EnableBatchProcessing

- RemoteChunkingManagerStepBuilderFactory: manager step
- RemoteChunkingWorkerBuilder: remote worker

# Partitionnement



# Partitionnement



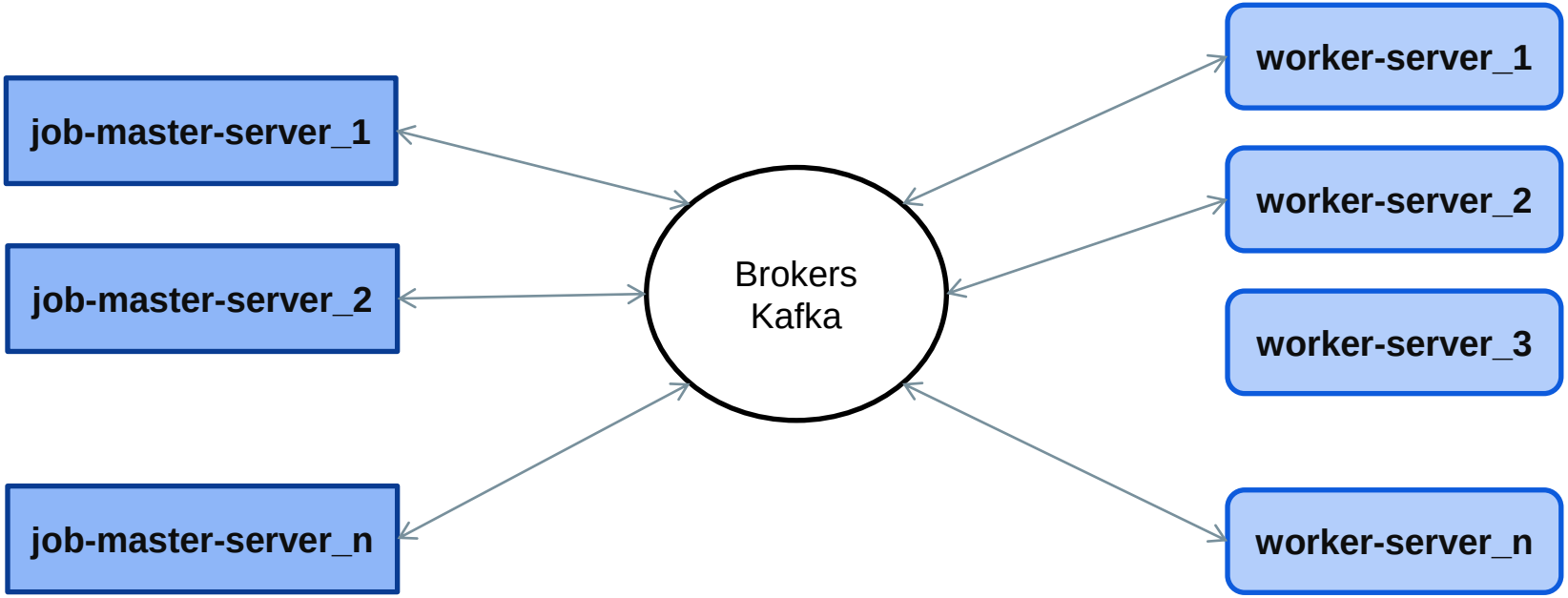
# Partitionnement

```
@Bean
public Step masterStep() {
    return stepBuilderFactory.get("masterStep")
            .partitioner("workerStep", partitioner())
            .partitionHandler(partitionHandler())
            .gridSize(2) // Nombre de serveurs de travail
            .build();
}
```

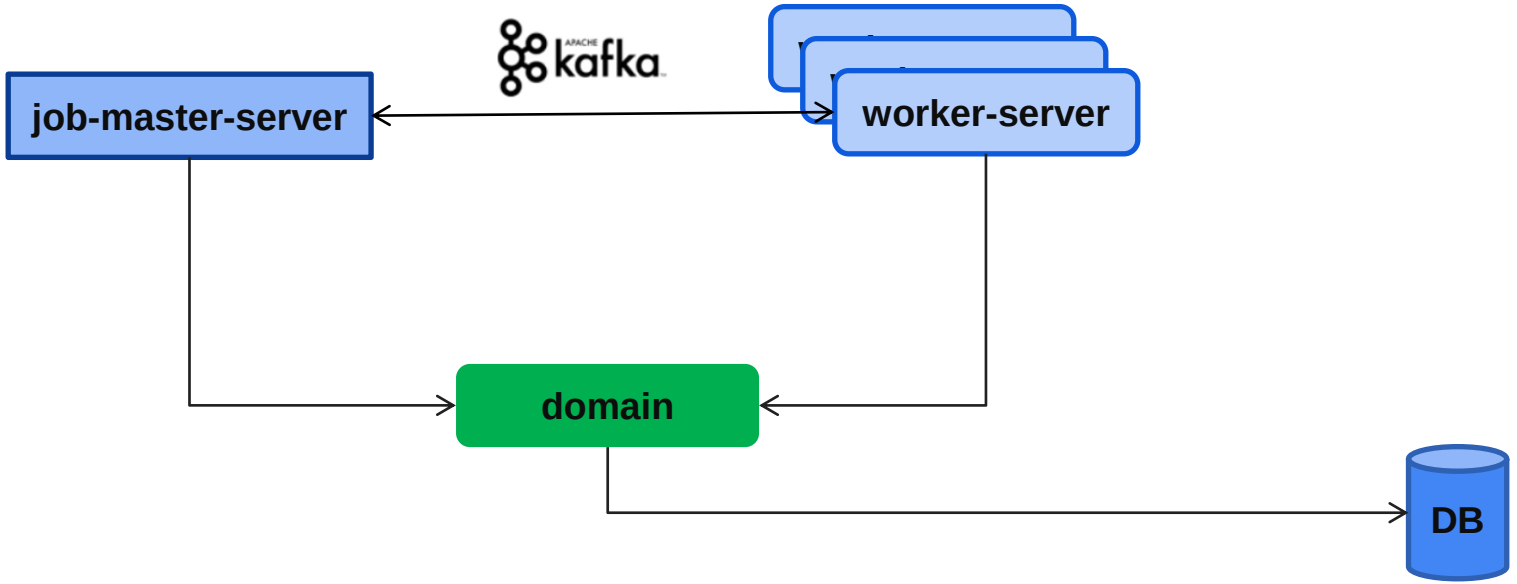
# Partitionnement

```
@Bean
public Step workerStep() {
    return stepBuilderFactory.get("workerStep")
            .<String, String>chunk(10)
            .reader(itemReader)
            .processor(itemProcessor)
            .writer(itemWriter).build();
}
```

# Fonctionnement avec Kafka



# 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