Introduction à l'événementiel

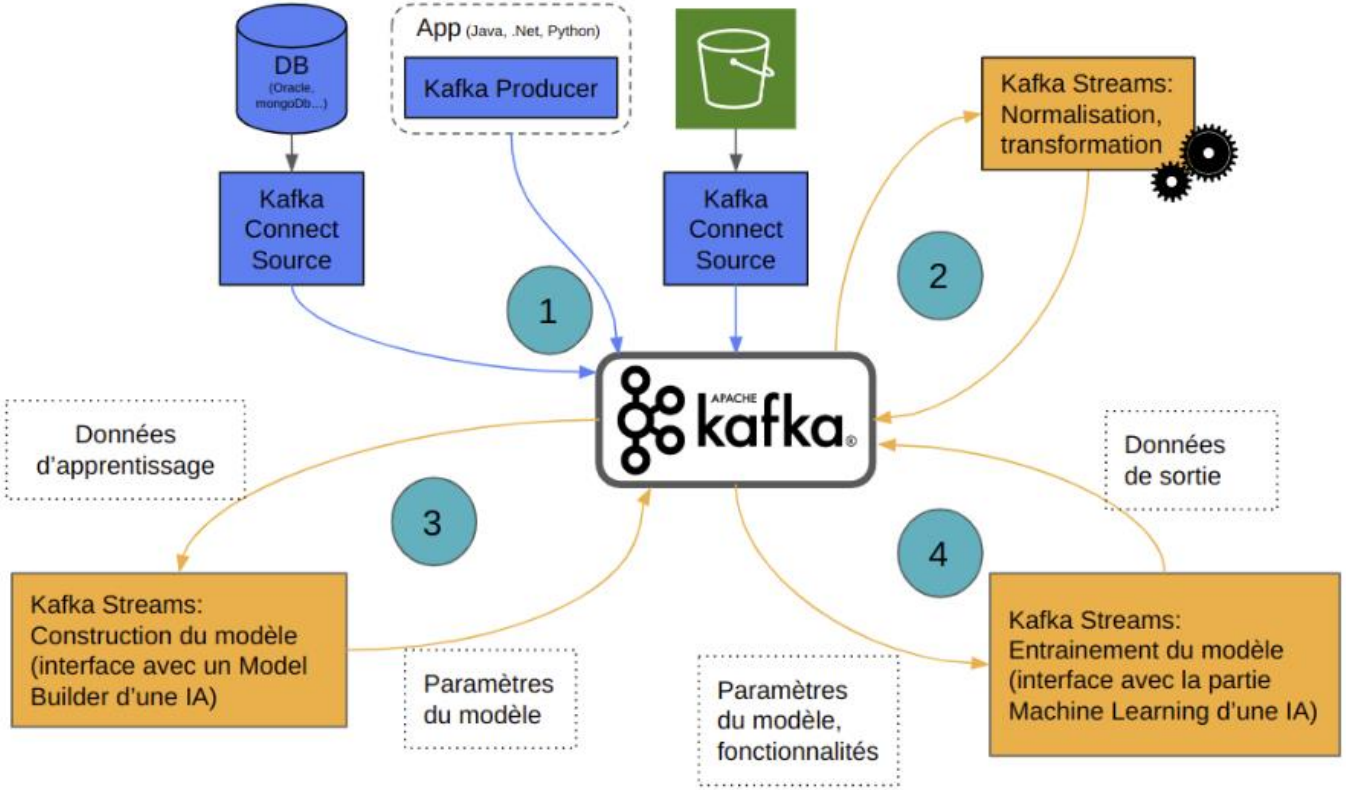
Spring Boot

Écosystème

Général

- Écosystème du cours / projet
  - Spring Boot
  - Spring Data
  - Spring Batch
  - Spring Cloud
  - Kafka

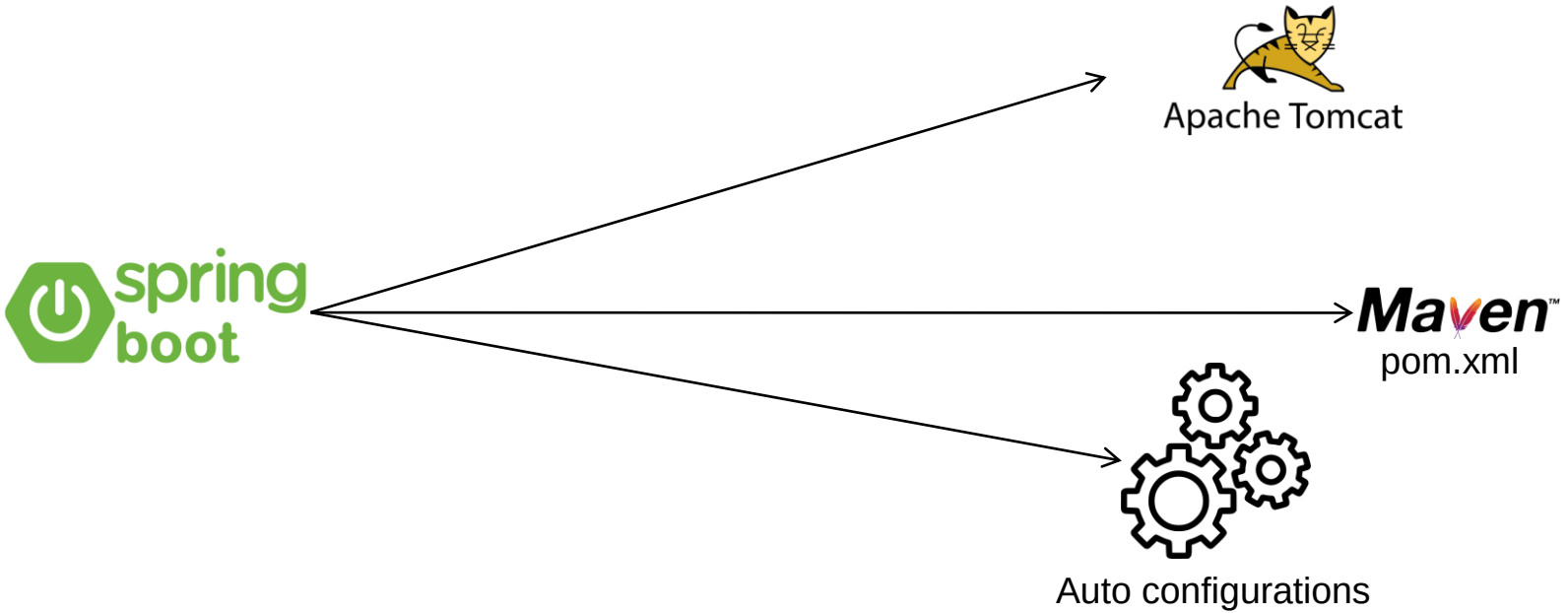
Kafka & l'IA


DB
(Oracle, mongoDb...)
App (Java, .Net, Python)
Kafka Producer
Kafka Connect Source
Kafka Connect Source
Kafka Streams: Normalisation, transformation
1
2
APACHE kafka
Données d'apprentissage
Données de sortie
3
4
Kafka Streams: Construction du modèle (interface avec un Model Builder d'une IA)
Kafka Streams: Entrainement du modèle (interface avec la partie Machine Learning d'une IA)
Paramètres du modèle
Paramètres du modèle, fonctionnalités

Spring

- Initialement juste un Framework Java
- Écosystème de projets Spring Framework
  - Spring Boot
  - Spring Data
  - Spring Batch
  - Spring Cloud

Spring Boot


Apache Tomcat
spring boot
Maven
pom.xml
Auto configurations

Spring boot

- **AOP** (Programmation par aspect) : Permet de créer dynamiquement des proxys s’exécutant à des évènements choisis dans le programme
- **Injection des dépendances** : Chaque bean créé est injectable via un fichier XML ou une annotation (injection dans le XML à travers @Autowired)
- **IOC** (Inversion de contrôle) : Un mécanisme qui facilite la mise en place des dépendances par l'injection automatique des objets (à travers des fichiers XML/annotations)

Injection de dépendance

```
public class EmailService {
   public void sendMessage(String msg, String str) {
        log.info("Email sent to " + str + " with message=" + msg);
   }
}
```

```
public class MyApplication {
   private EmailService email = new EmailService();

   public void processMessage(String msg, String rec) {
      // do some validation, add some logic, etc.
      this.email.sendEmail(msg, rec);
   }
}
```

Injection de dépendance

```
public interface MessageService {
   boolean sendMessage(String msg, String rec);
}
```

```
@Service
public class EmailService implements MessageService {
   public void sendMessage(String msg, String rec) {
      log.info("Email sent to "+rec+ "with message="+msg);
   }
}
```

```
@Component
public class MyApplication {
   @Autowired
   private MessageService service;

   public void processMessage(String msg, String rec) {
      // do some validation, add some logic, etc.
      this.service.sendEmail(msg, rec);
   }
}
```

Injection de dépendance

```
public interface MessageService {
   boolean sendMessage(String msg, String rec);
}
```

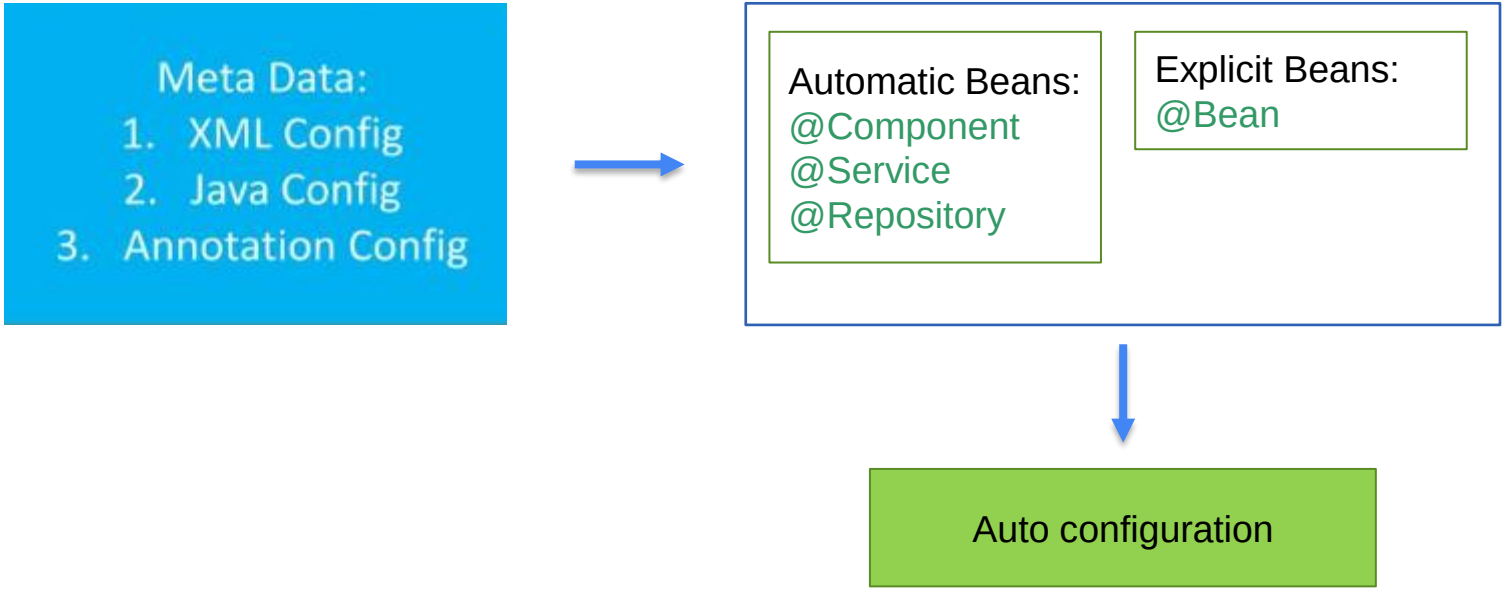
```
public class EmailService implements MessageService {
   public void sendMessage(String msg, String rec) {
       log.info("Email sent to "+rec+ " with message="+msg);
   }
}
```

```
public class SmsService implements MessageService {
   public void sendMessage(String msg, String rec) {
       log.info("SMS sent to "+rec+ " with message="+msg);
   }
}
```

```
@Configuration
public class MessageConfiguration {
  @Bean(“EmailService”)
  public MessageService getMessageService() {
     return new EmailService();
} }
```

```
@Component
public class MyApplication {
  @Autowired(“EmailService”)
   private MessageService service;
  /* … */
}
```

Spring Ioc


Meta Data:
1. XML Config
2. Java Config
3. Annotation Config
Automatic Beans:
@Component
@Service
@Repository
Explicit Beans:
@Bean
Auto configuration

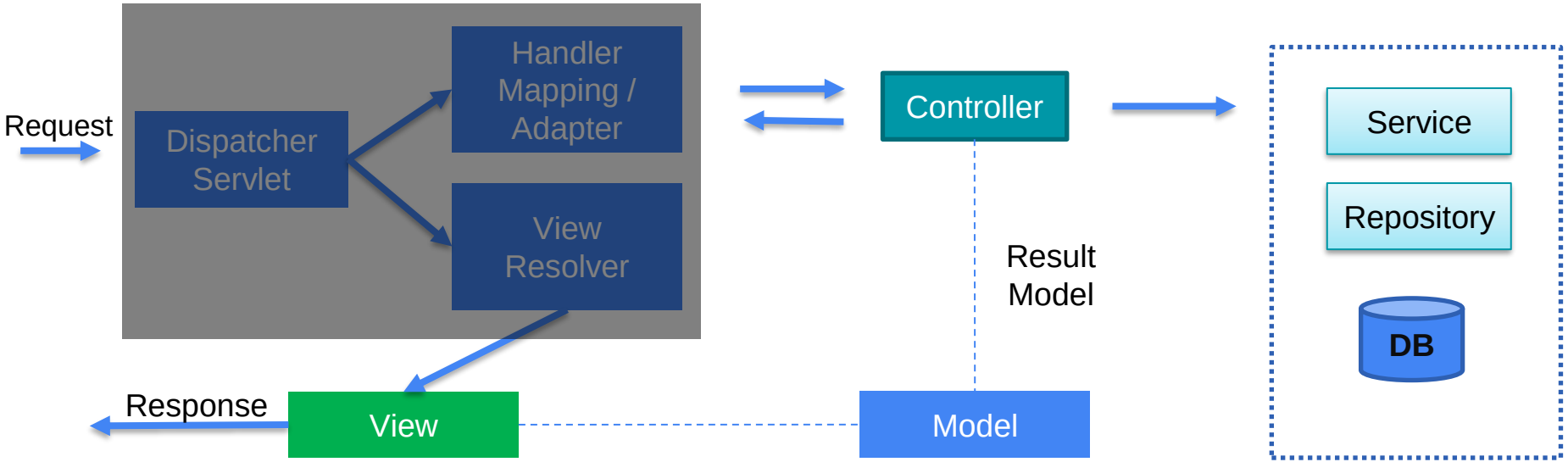
Inversion de contrôle (IOC)

IOC & Injection de dépendance

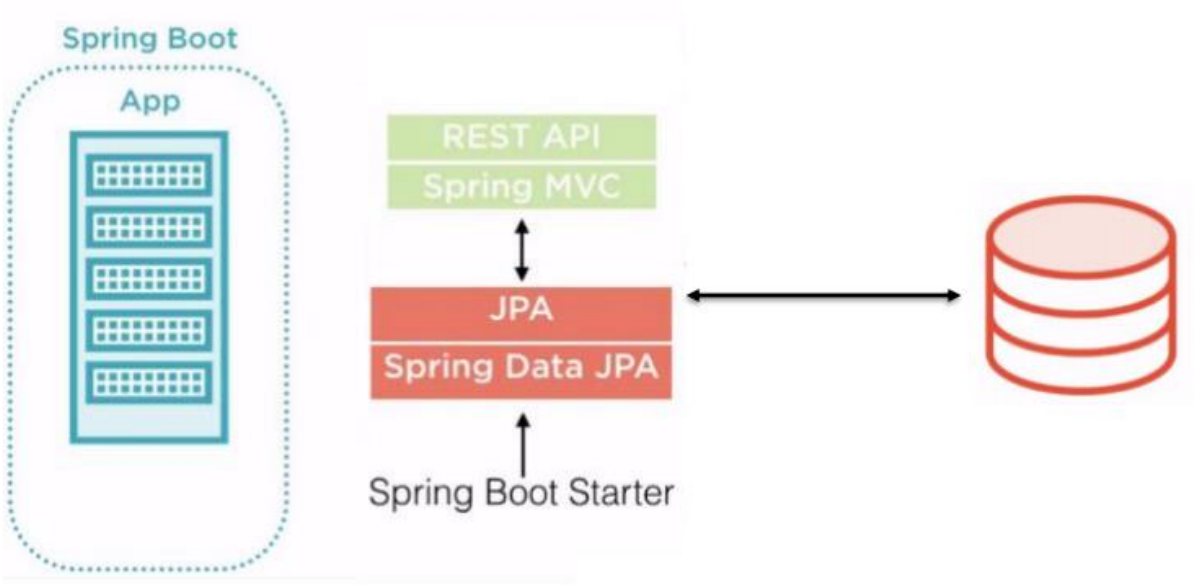
```
// Tradiotional Way
public class ExampleService implements ExampleService {
   private ExampleRepository repo = new ExampleRepository ();
}
```

```
// Dependency Injection
@Service
public class ExampleService implements ExampleService {
  @Autowired
   private ExampleRepository repo;
}
```

MVC / Architecture



Persistence - JPA



Persistence - JPA

- Entity
- Repository
- Service
- Controller

- @Entity @Table
- @Repository
- @Service
- @RestController @RequestMapping

Entité

```java
@Entity
@Table(name = "entity")
@Getter @Setter @ToString
@NoArgsConstructor
public class Entity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(name = "label")
    private String label;
}
```

Repository

- Couche d’accès aux entités
  - JpaRepository<T, ID>
- Génération automatique de requêtes (méthodes pré-implémentées)
  - count()
  - delete() / deleteAll()
  - save() / saveAll()
  - findById() / findAll() / findAllById()

```
@Repository
public interface EntityRepository extends JpaRepository<Entity, Integer> {
  Entity getByLabel(final String str);
}
```

APIs Rest

- Annotations de base
  - @Controller
    - @RestController : @Controller with @ResponseBody
  - @RequestMapping
    - @*Mapping (@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, etc.)

```
@RestController
@RequestMapping(value = "/entity", produces = JsonUtils.MEDIA_TYPE_JSON_UTF8)
public class EntityApiController {
   @GetMapping("/getAll")
   public List<Entity> getAll() {
      /***/
   }
}
```

APIs Rest

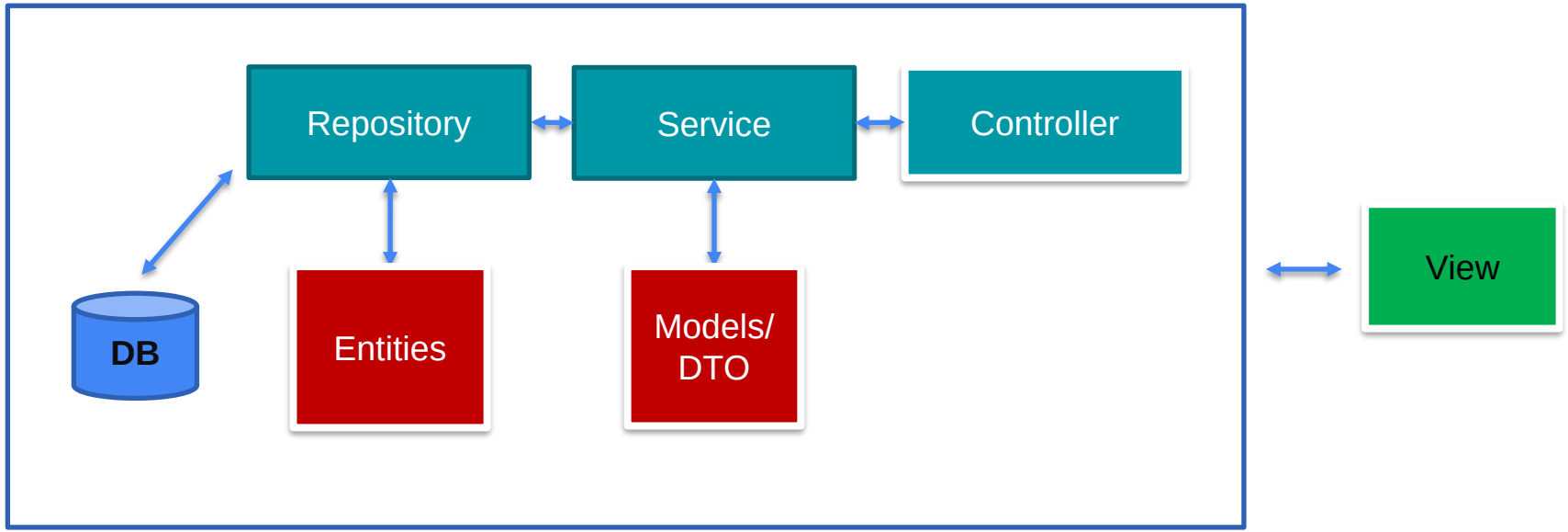
- @RequestParam
  - /getByLabel?label=toto

```
@GetMapping("/getByLabel")
public Entity getByLabel(@RequestParam(value = "label", defaultValue = "label_entity1") String input) {
  return _entityRepository.getByLabel(input);
}
```

- @PathVariable

```
@GetMapping("/getByid{id}")
public Entity getByLabel(@PathVariable(value = "id") final Integer input) {
  return _entityRepository.getOne(input);
}
```

Architecture



Lombok

- Librairie Java pour générer du bytecode
  - + plugin IntelliJ
- Plus besoin d’écrire de code sans valeur ajoutée
  - Getter/setter
  - Constructeurs
  - Hashcode/equals
  - Logger

Lombok - getter / setter

- @Getter / @Setter
  - Sur un attribut
  - Sur une classe pour tous les attributs
  - public par défaut
    - Visibilité surchargeable :
    - @Setter(AccessLevel.PROTECTED)

```
public class GetterSetterExample {
  @Getter @Setter
  private int age = 10;
  @Setter(AccessLevel.PROTECTED)
  private String name;
}
```

```
public class GetterSetterExample {
  private int age = 10;
  private String name;

  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  protected void setName(String name) {
    this.name = name;
  }
}
```

Lombok - constructeur

- @NoArgsConstructor
  - sans argument
- @RequiredArgsConstructor
  - avec tous les attributs finaux ou @NonNull
- @AllArgsConstructor
  - avec tous les attributs

```
@RequiredArgsConstructor(staticName = "of")
@AllArgsConstructor(access = AccessLevel.PROTECTED)
public class ConstructorExample<T> {
  private int x, y;
  private T description;
}
```

```
public class ConstructorExample<T> {
  private int x, y;
  private final T description;

  private ConstructorExample(T description) {
    this.description = description;
  }
  public static <T> ConstructorExample<T> of(T description) {
   return new ConstructorExample<T>(description);
  }
  protected ConstructorExample(int x, int y, T description) {
                this.x = x;
                this.y = y;
                this.description = description;
  }
}
```

Lombok – equals & hashcode

- @EqualsAndHashCode

```
@EqualsAndHashCode
public class EqualsAndHashCodeExample {

  private transient int transientVar = 10;
   private int id;
   private String name;
   @EqualsAndHashCode.Exclude
   private double score;

}
```

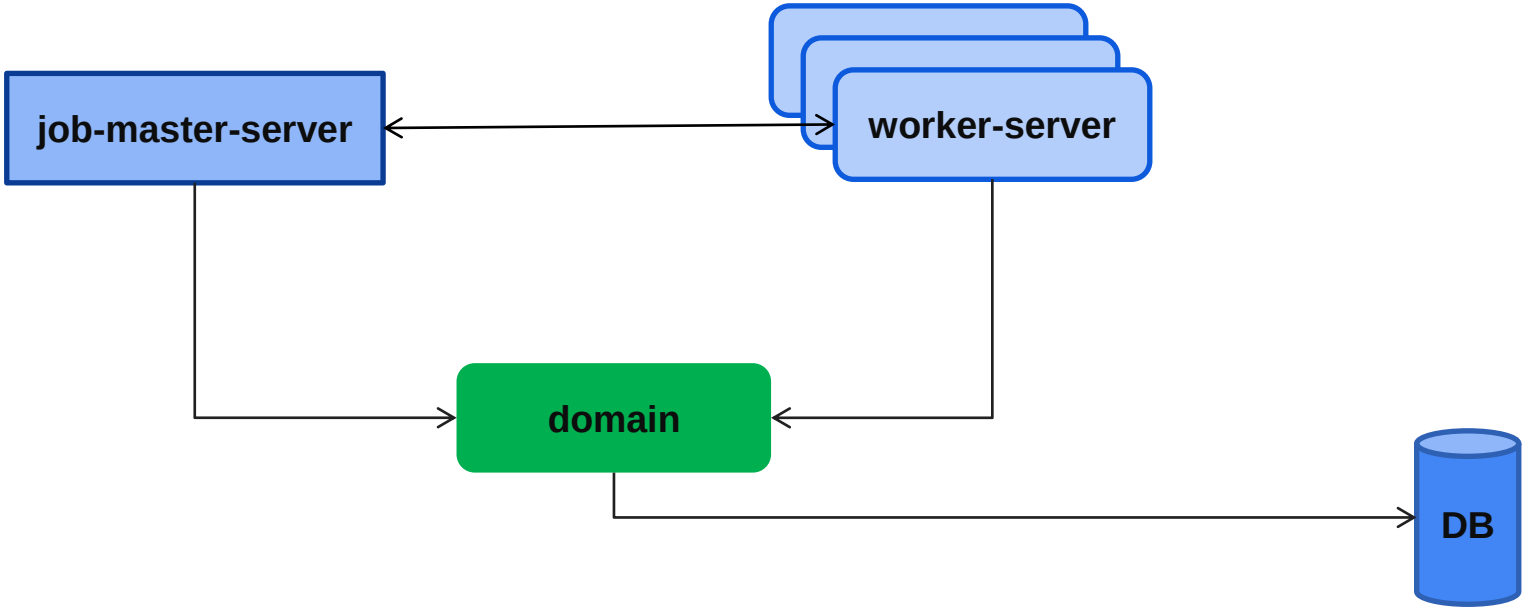
Modalités d'évaluation

Rendu final :

- Code source
- Soutenance (avec démo)
- Rapport (3 / 4 pages)

Deadline: vendredi 26 mai 20:00

Architecture



Suite - Spring Batch Kafka