DataLifecycle Management

Implement a complete solution by mixing SQL & NoSQL solutions

2021-03-08

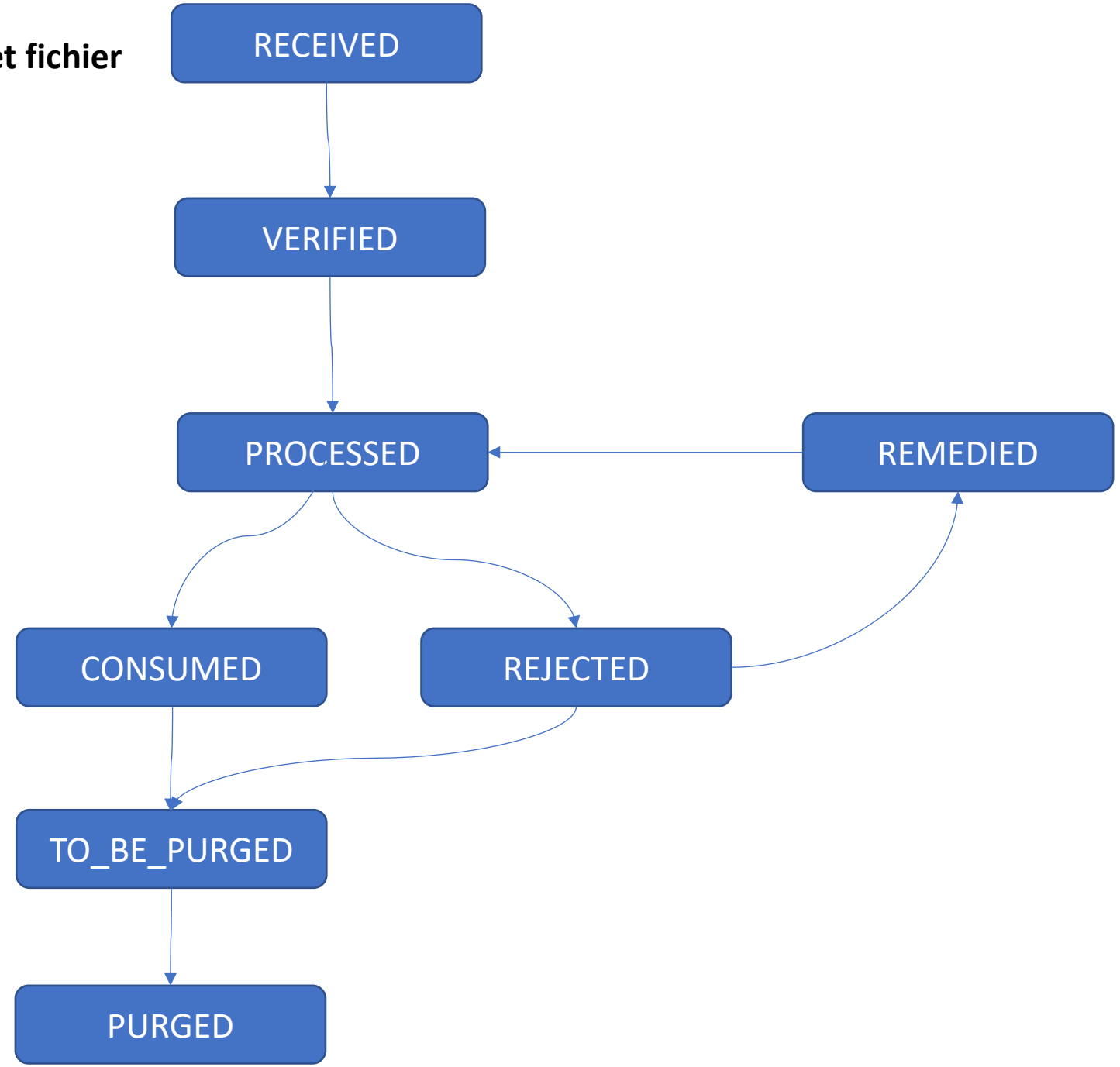
Sujet : Créer une solution d'ingestion et d'analyse de donnés de cycle de vie

Utilisation : Base relationnelle, base NoSQL MongoDB & base Redis

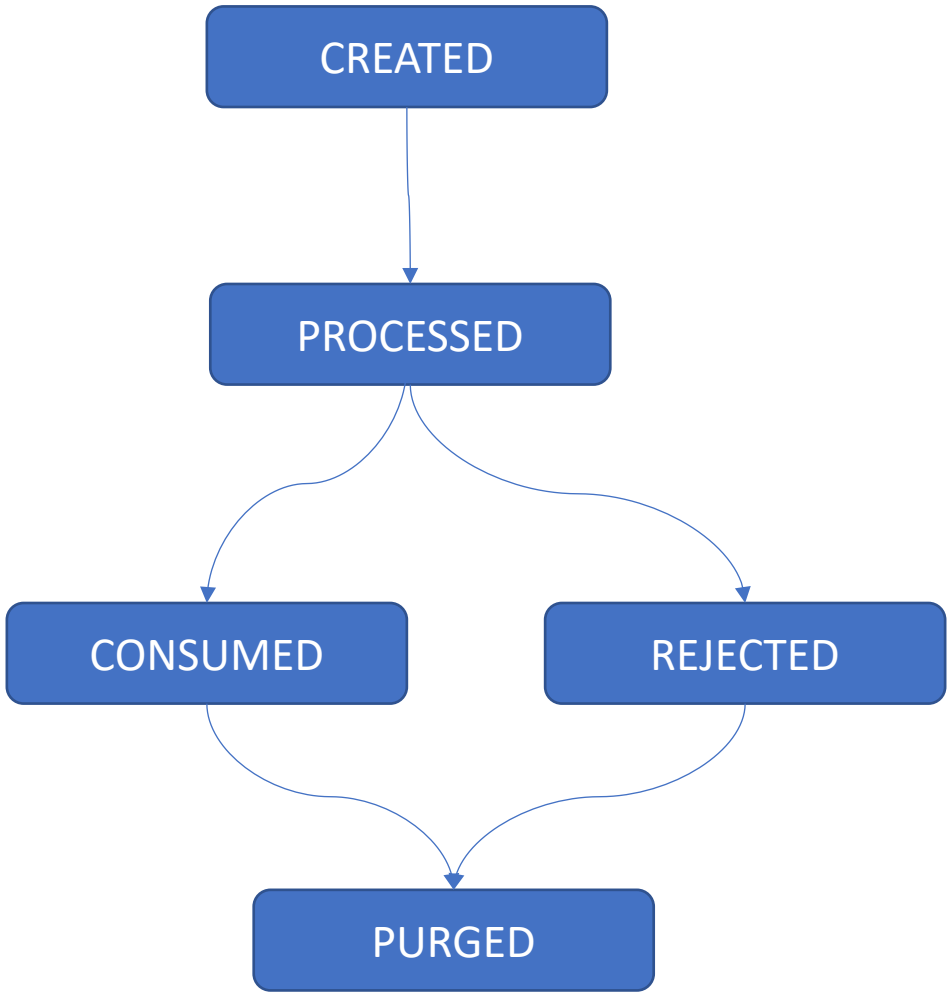
Modalités

- Constituer les groupes de 3 personnes maximum (1 personne est accepté)
- Rendre l’application après le dernier cours
- Rendre le TP sur GitHub
- Envoyer un mail avec toutes les indications à gregory.boissinot@gmail.com

Exemple de cycle de vie d’un objet fichier



Exemple de cycle de vie d’un objet element d’un fichier



Data Lifecycle Event (*sample*)


```
{
"id":"c3316565-7428-48a8-91e3-34afaf5e4cfe",
"event-type": "TRACE_EVENT",
"occurredOn":"2021-03-07T21:12:12.039020",
"version":1,
"graph-id":"file-states",
"nature":"raw-file",
"object-name":”File025",
"path":"[RECEIVED, VERIFIED, PROCESSED, CONSUMED]”
}
```

Réception de Data Lifecycle Event (*sample*)

temps

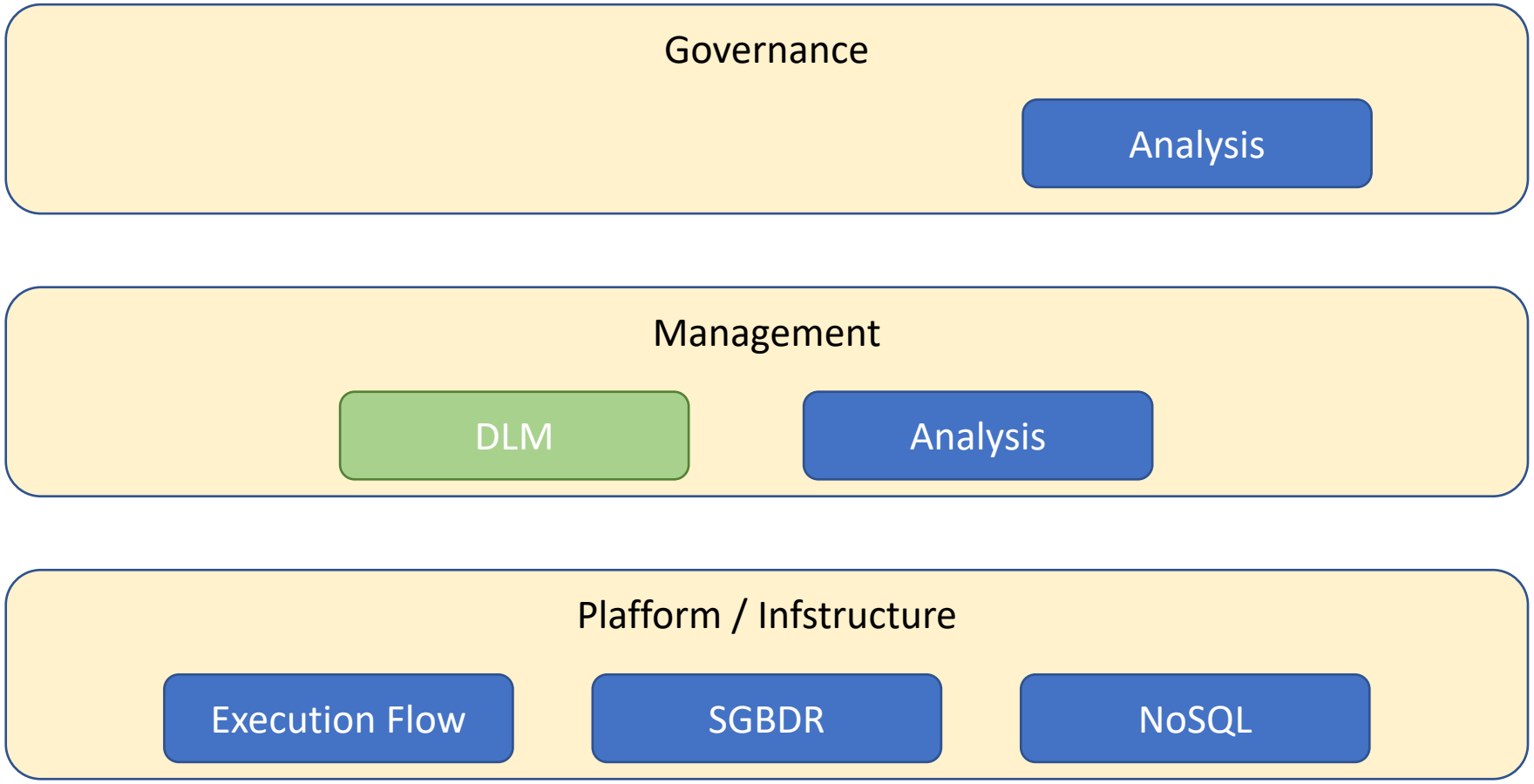
{
"id":"c3316565-7428-48a8-91e3-34afaf5e4cfe",
"event-type": "TRACE_EVENT",
"occurredOn":"2021-03-07T21:12:12.039020",
"version":1,
"graph-id":"file-states",
"nature":"raw-file",
"object-name":"**File025**",
"path":"[RECEIVED, VERIFIED, PROCESSED, REJECTED, REMEDIED]"
}

{
"id":"c3316565-7428-48a8-91e3-34afaf5e4cfe",
"event-type": "TRACE_EVENT",
"occurredOn":"2021-03-07T21:12:12.039020",
"version":1,
"graph-id":"file-states",
"nature":"raw-file",
"object-name":"**File025**",
"path":"[PROCESSED, CONSUMED]"
}

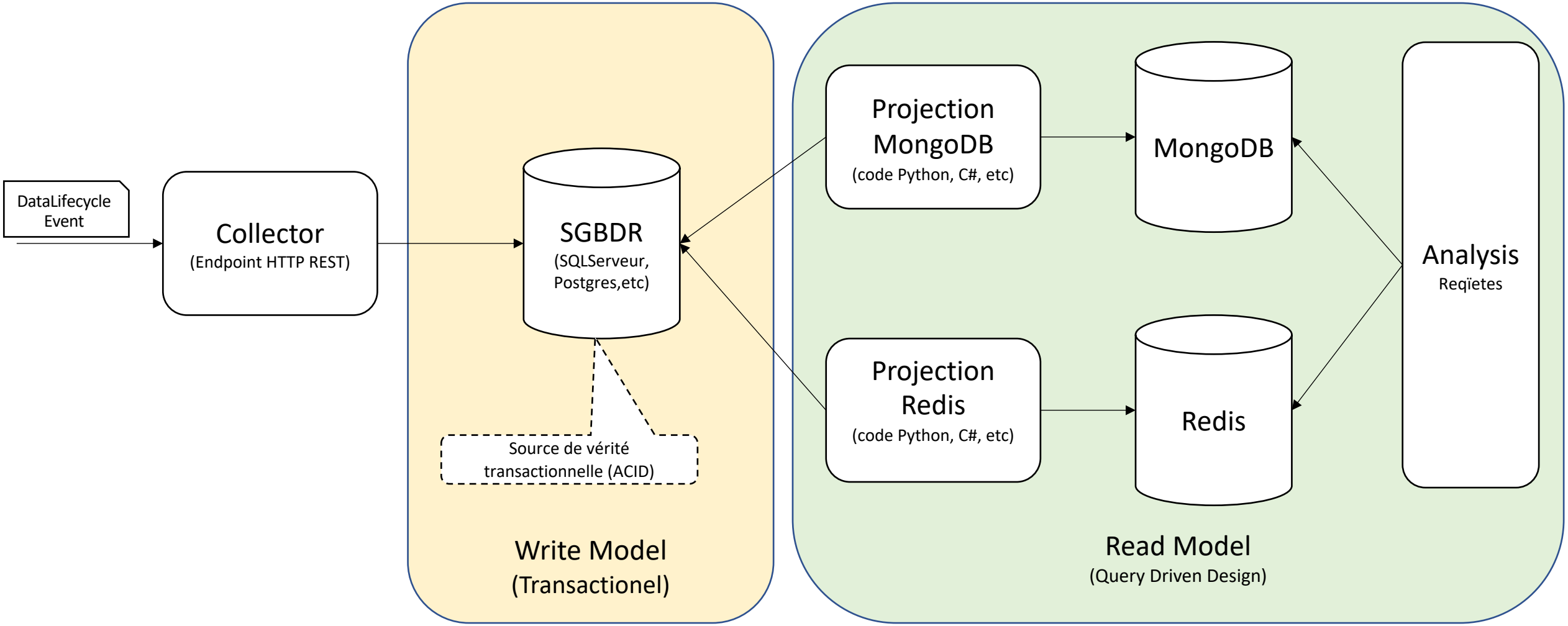
{
"id":"c3316565-7428-48a8-91e3-34afaf5e4cfe",
"event-type": "TRACE_EVENT",
"occurredOn":"2021-03-07T21:12:12.039020",
"version":1,
"graph-id":"file-states",
"nature":"raw-file",
"object-name":"**File025**",
"path":"[TO_BE_PURGED, PURGED]"
}

Le cycle de vie est fragmenté par plusieurs blocs d'exécution
Une mécanique d'assemblage est nécessaire pour reconstituer tout le cycle de vie d'un objet donnée

Data Lifecycle Management (DLM)



Cinématique de la solution d'ingestion



Use cases à implémenter (sous forme de requêtes)

PARTIE 1

- Récupérer le cycle de vie parcouru (la liste des status d'un objet donné)
- Compter le nombre d'objets par status
- Compter le nombre d'objets par status sur la dernière heure
- Compter le nombre d'objets respectant l'intégrité du graphe du cycle de vie

Use cases à implémenter (sous forme de requêtes)

Sujet: Combiner plusieurs graphe de données de cycle de vie

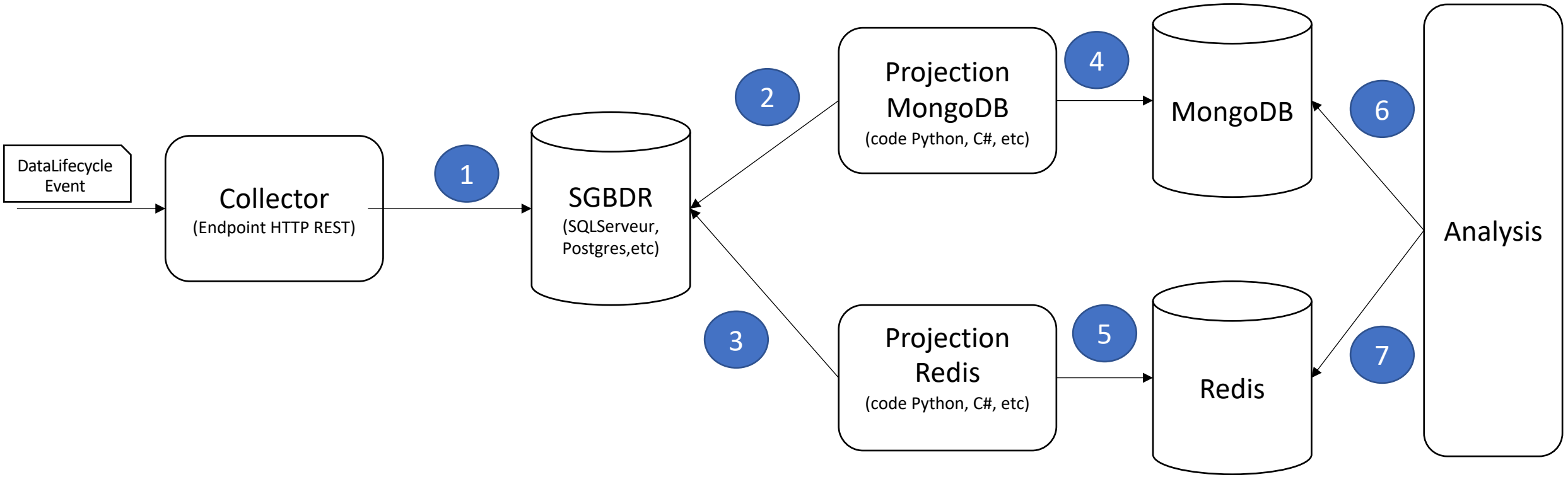
PARTIE 2
(en option)

- Pour un objet de nature fichier, compter le nombre d'objets dans ce fcihier
- Reprendre l'ensemble des cas d'utilisation de la partie 1 et ajouter de la traçabilité (sous forme de métadonnées) avec l'objet parent

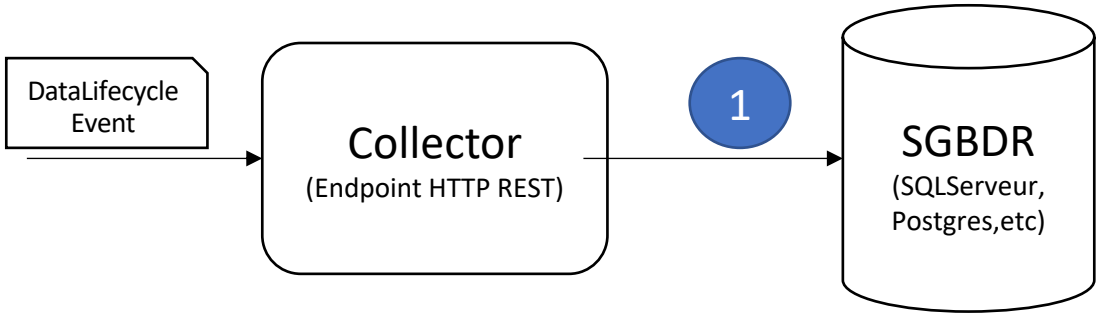
Préconisations

- Favoriser l’usage d’une technologie de containerisation tel que **Docker** pour vos tests
- Commencer par un premier modèle de vos différentes données pour vos différents cas d’utilisations

Cinématique & ordonnancement



Indications Flux numéro 1

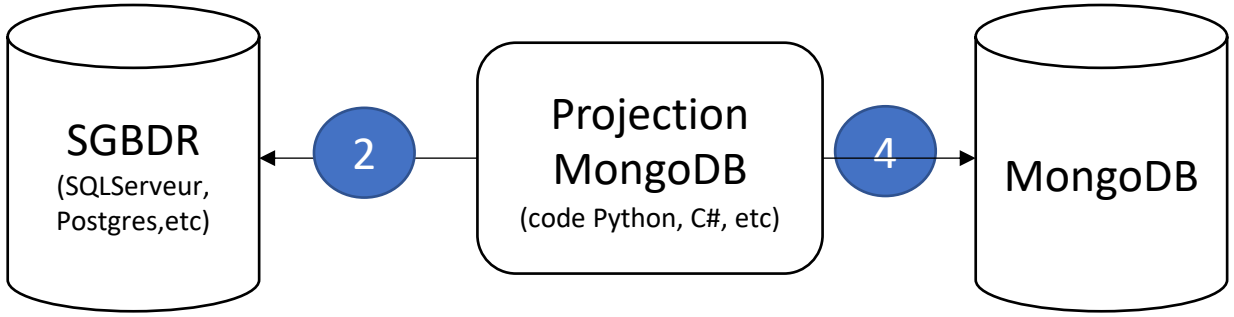


- Le REST Controller reçoit une requête HTTP comprenant un flux JSON constitué de 1 ou plusieurs event de cycle de vie
- Le REST Controller est dit un “pass-plat”
- Les différents champs de l’evenement sont stockés dans la base de données
- Une seul table relationnel est utilisé
  - Chaque ligne comprend un événement du cycle de vie
- Il faut créer un injecteur de données (ex: script qui injecte 10 000 données d’évent)
- Autant de ligne dans le SGBDR que d’event reçu

- <u>Points d’attention</u>
  - Attention à la gestion des dates
  - Distinguer les différentes dates : date d’ingestion de la données dans le système, date de la création de l’événement de cycle de vie
  - Utiliser le bon type de données pour représenter les données associées à l’évenement (tout est possible, y compris type JSON PostgreSQL)

- <u>Détails d’implémentation</u>
  - Utiliser une méthode HTTP POST
  - En option, renvoyer un status code 201 avec l’id de l’event crée dans le système

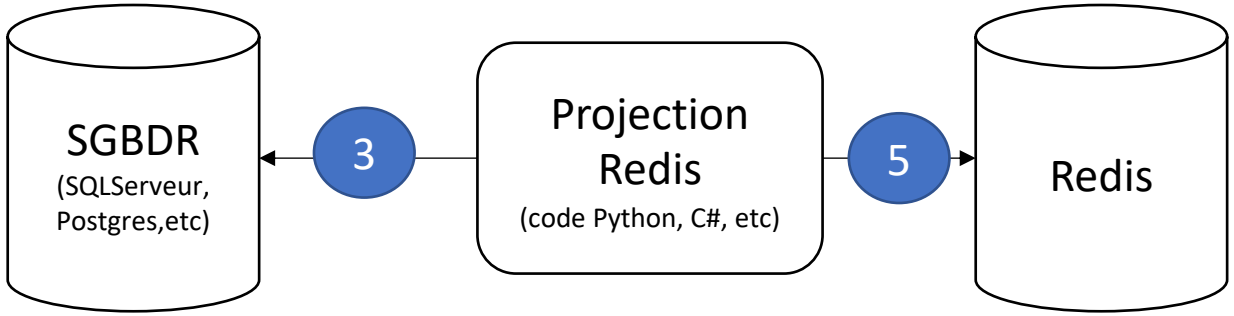
Indications Flux numéro 2 et 4



- La projection est un programme de code
- Tout langage est accepté (du shell/bash à du Java/C# en passant par Python)
- Fonctionnement
  - La projection comporte un scheduler (ex: exécution d'un processus à fréquence régulière comme par exemple toute les 5 minutes)
  - Scheduler
    - Interrogation (Polling) des nouvelles données dans la base de données relationnelle
      - Mécanisme de sauvegarde d'un curseur
      - Recupération du ResultSet des nouvelles données
      - Application d'un mapper entre les données relationnelles et les données au format MongoDB
      - Insertion (ou requête de mise à jour de données) dans MongoDB

- Exemple de requête de mise à jour
  - MongoDB : un objet est un document et l'*history* est un attribut de type liste de ce document

Indications Flux numéro 3 et 5



- Reprise des principes de la projection de MongoBD

- Exemple de requête de mise à jour
  - Redis : une clé de type compteur qui comporte le nombre d'objet qui sont rejetés
    - *Sémantique de la clé*
    - *Utilisaiton de l'API REDIS associé*

```
127.0.0.1:6379> set states:rejected 1
OK
127.0.0.1:6379> incrby states:rejected 5
(integer) 6
127.0.0.1:6379> get states:rejected
"6”
```

```
127.0.0.1:6379> hmset files:F025:states CONSUMED 1 REJECTED 0
OK
```

Indications Flux numéro 6 et 7

- Fournir les requêtes corresponsant aux différents cas d’utilisations