

Base de données multimédia

&

Bases de données documentaires et distribuées

Justine Coutelier

Julie Courgibet

2020-2021

Sommaire

Introduction

1. Objectif
2. Enjeux
3. Usages
4. Planning

I. Description de la solution

1. Architecture
2. Récupération des données (UIPath)
3. Insertion des données (Python)
4. Analyse des données
 - a. Requêtes : Sélection
 - i. Liste des collections de la base de données et nombre de documents
 - ii. Liste des vendeurs d'une collection (exemple : foire)
 - b. Requêtes : plus complexes
 - i. Moyenne des prix des voitures
 - ii. Map-Reduce : Liste des marques des voitures
 - iii. Moyenne des prix des voitures en fonction des marques
 - iv. Moyenne des prix des voitures en fonction de l'âge de la voiture
 - v. Nombre de vêtements par catégorie
 - vi. Prix des vêtements par catégorie
 - vii. Type de vêtements vendus
 - c. Visualisation des données
 - i. Distribution des prix par catégorie

II. Description des livrables

III. Analyse de la technologie

1. MongoDB : Base de données noSQL documentaire
2. Elasticsearch : Base de données noSQL documentaire (moteur de recherche Lucene)
3. Volume et scalabilité
 - a. Replicaset : Tolérance aux pannes et disponibilité des données
 - i. Stratégie de déploiement
 - ii. Création d'un ReplicaSet
 - iii. Résistance aux pannes
 - b. Sharding : Scalabilité horizontale
 - i. Prérequis
 - ii. Structure du cluster
 - iii. Réalisation du cluster
 - iv. Ingestion de données dans le cluster

Conclusion

Introduction

Ce projet consiste à récupérer les données d'un site de vente en ligne et les organiser dans une base de données.

On va récupérer les données du site leboncoin (LBC). Des utilisateurs français postent des annonces en ligne pour vendre des objets ou des services. Il y a une énorme diversité d'objets en vente qui sont catégorisés :

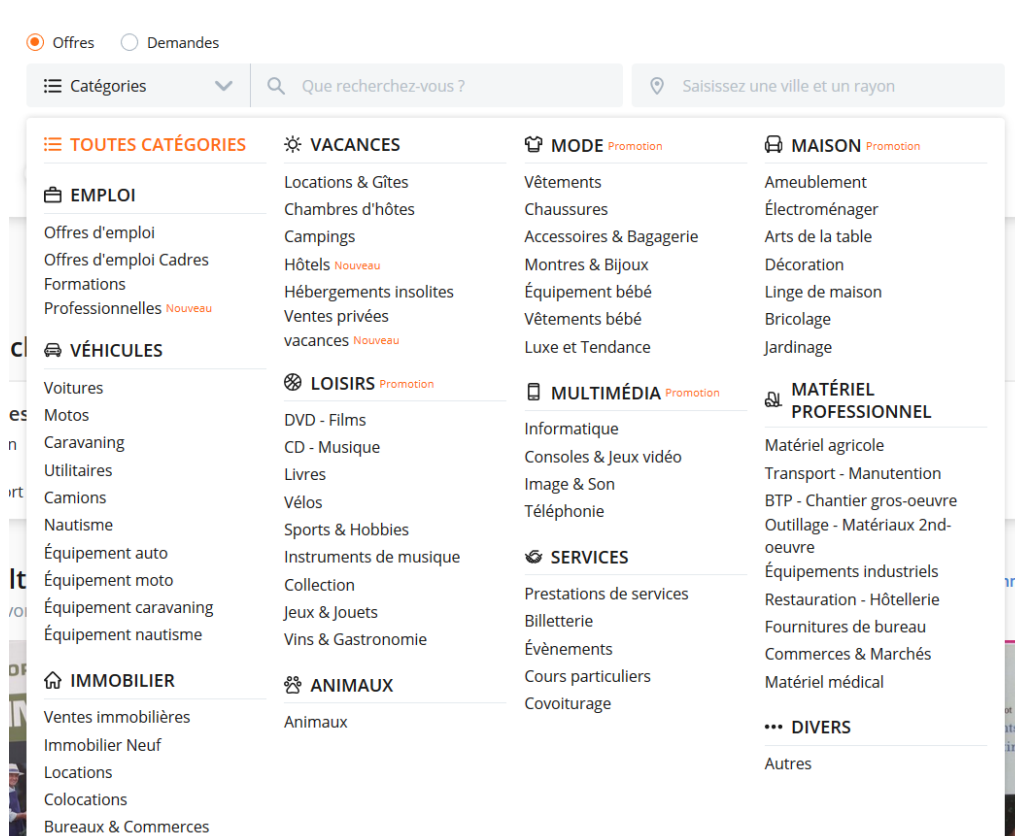


Figure 1 : catégorie des offres en ligne

Comme on peut le voir, le site peut proposer de poster des offres (objets ou services contre de l'argent) ou de poster des demandes (les gens recherchent des offres ou services).

Le jour de la réalisation de ce rapport, 38 939 826 annonces étaient en ligne.

LBC est une base de données protégée, producteur et exploitant de sa propre base [1]. Il n'est donc pas possible d'extraire les données de façon simple. Pour cela, il va falloir interroger les pages internet du site LBC et récupérer les données automatiquement. LBC a une architecture basée sur 500 serveurs [2]. Au début du site, LBC utilisait MySQL, puis avec la montée en charge, LBC a migré sur PostgreSQL [2]. Pour l'analyse des données, LBC utilise Amazon Cloud et développe des outils internes à base de solutions open source.

1. Objectif

Ce projet va s'inscrire dans un projet plus global de Big data. Il s'agit de la première brique qui consiste à récupérer, traiter et organiser les données affichées sur chaque annonce en ligne. L'accès à ces données via une base de données permettra ainsi d'interroger la BDD et d'en tirer de la valeur (statistiques, Machine Learning, savoirs, etc).

L'objectif de cette brique est donc de mettre à disposition dans une base de données toutes les annonces accessibles en temps réel. Cette BDD sera enrichie tous les jours avec les nouvelles annonces et mise à jour avec un suivi des annonces.

2. Enjeux

Cette brique est essentielle pour comprendre les comportements d'achat et la réalisation de statistiques (prix de vente, catégories en forte demande, etc). En effet, pour acquérir du savoir dans ce domaine, il est nécessaire de le tirer de données et donc d'accéder à cette donnée. Il est évident que LBC est propriétaire de ces données et ne les donne pas gratuitement. Il est donc obligatoire de récupérer ces données. La réalisation de cette brique nous permet donc de gagner de la donnée, qui est aujourd'hui l'or numérique.

La donnée permet en effet de créer de la valeur [3]. L'enjeu est donc le gain en données pour créer de la valeur.

D'un point de vue plus humain, on y gagne aussi en compétences personnelles :

- Gestion de projet :
 - Méthode Agile ;
 - Travail en équipe ;
 - Pouvoir mener et terminer un projet en le définissant et en présentant un produit fini.
- Compétences techniques :
 - Langage de programmation orienté objet (ex : Python)
 - Base de données noSQL (ex : MongoDB)
 - Statistiques (ex : Python et librairies pandas, numpy, maths, etc ou R).

3. Usages

Cette brique s'inscrirait dans un projet plus global qui souhaite utiliser les données recueillies pour monter un modèle de Machine Learning (IA). Cependant, ces données sont un puits de valeur immense qui permettent de répondre à de petites interrogations comme à de plus grandes questions.

Pour les petites interrogations (statistiques), il suffira d'interroger la BDD via MongoDB :

- Durée de vie des annonces ;
 - Cette question peut être déclinée selon plusieurs critères :
 - Catégorie ;
 - Géographie (région, département, ville) ;
 - Utilisateur ;

- Prix ;
- Qualité de l'annonce (* voir la partie sur les questions de ML).
- Domaines et objets les plus vendus ou les plus demandés ;
 - Il y a une valeur à tirer (investissements).
- Définir l'intérêt des photos et des descriptions (textes) pour la vente des objets.
- Définir les comportements de vente :
 - Nombre d'annonces par utilisateur
 - Diversité des objets vendus ;
 - Prix des objets (Objets plus ou moins cher que la moyenne).
- Tirer de la valeur sur la géographie :
 - Prix selon la géographie ;
 - Nombre d'annonces selon la géographie ;
 - Catégories selon la géographie.

Il y a d'autres très nombreuses questions statistiques à se poser, qui peuvent mener à des questions plus globales et complexes.

Ainsi pour les interrogations plus complexes, cela demandera de développer un modèle de ML :

- Définir la qualité d'une annonce :
 - Traitement du texte – NLP Natural Language Processing sur le texte de description de l'annonce ;
 - Traitement d'image – Computer Vision sur les images postées.
- Définir l'intérêt d'une annonce.
- Repérer la fraude :
 - Prix ?
 - Photo « volées ».
 - Profil utilisateur suspect

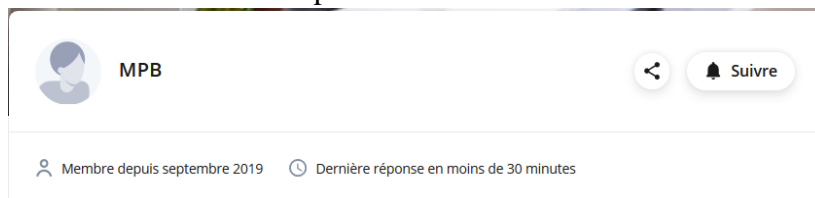


Figure 3 : Exemple d'un profil utilisateur

On peut récupérer le nom, l'identifiant (via l'URL), la date d'inscription, le nombre d'annonces).

4. Planning

La gestion du planning a été réalisée avec l'outil Trello. Nous avons découpé le projet en plusieurs US (user stories) :

- Robot –Scrapping data :
 - Automatisation de la récupération des données
 - Récupération des données sur LBC pour catégorie véhicules
 - Modifier script automatisation de récupération des données LBC pour d'autres catégories
 - Récupérer données des vendeurs : scripts UiPath pour automatiser la recherche des pages internet

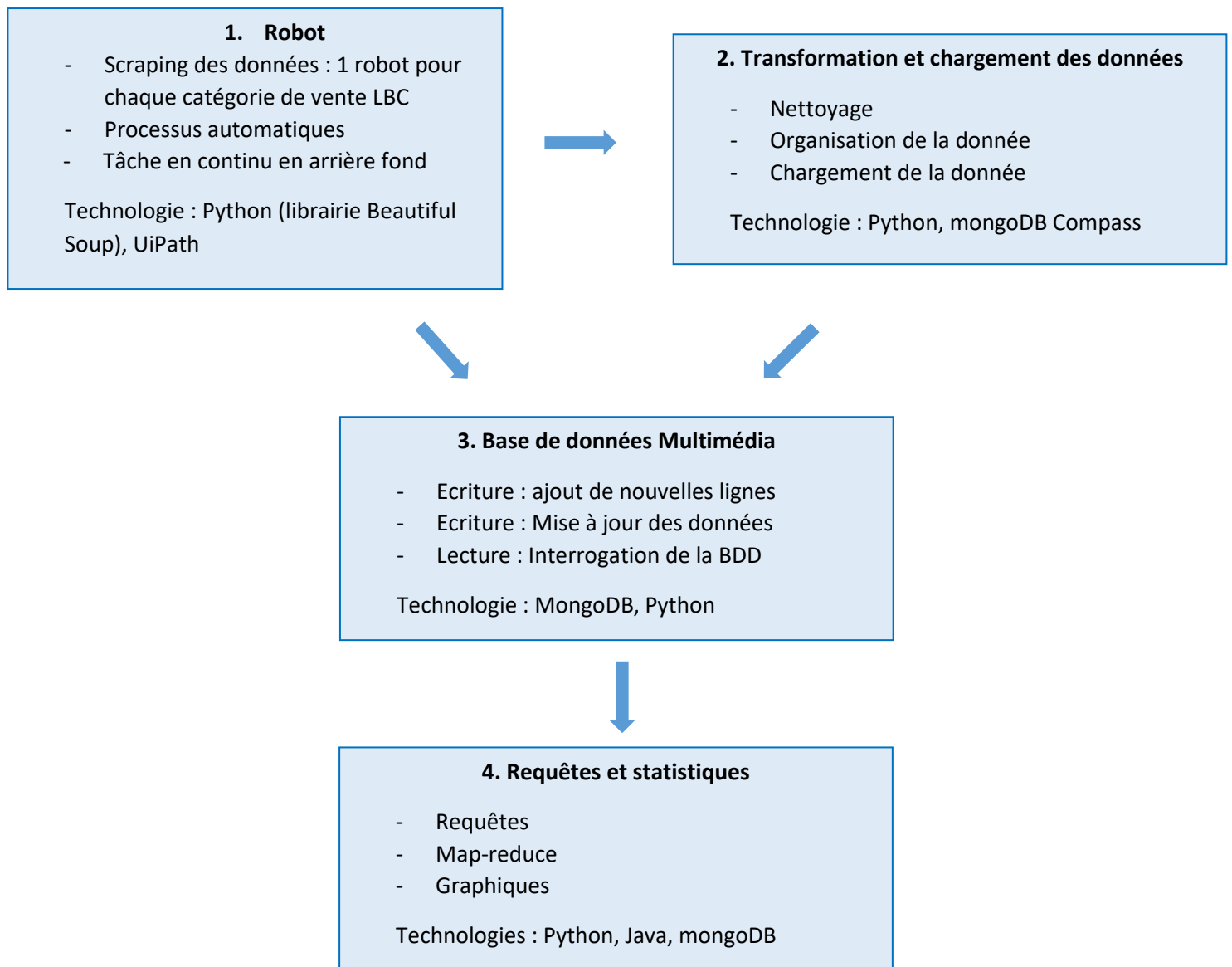
- Récupérer données des vendeurs : script Python
- Faire tourner le robot pour récupérer données
- Automatisation pour mise à jour des données – interrogation du site LBC partir des données enregistrées (URL)
- Transformation et chargement des données :
 - Nettoyer les données (fichier html)
 - Organisation de la donnée pour le chargement dans la base
 - Chargement des données dans mongoDB
 - Mise à jour des données de la base mongoDB (annonces encore en ligne ?)
- Base de données mongoDB :
 - Installer mongoDB sur nos 2 postes
 - Vérifier que mongoDB fonctionne
 - Créer d'autres collections dans la base mongoDB
 - Interrogation de la BDD – requêtes
 - Interrogation type sélection
 - Interrogation type agrégation
- Statistiques
 - Map-reduce
 - Faire des statistiques
 - Graphiques
- Scalabilité
 - ReplicaSet
 - Sharding
- Rendu
 - Faire un trello
 - Faire des vidéos de démo
 - Faire un powerpoint
 - Captures d'écran – requêtes
 - Enregistrer un vrai cas- user story

I. Description de la solution

1. Architecture

Ce projet est réalisé comme un POC (Proof of Concept) et fonctionnera sur nos machines physiques (stockage local). Les services Cloud ne seront pas utilisés car payants. Cependant, en collaboration avec le projet de Nicolas, on a aussi pu stocker nos données sur le cloud. Le cloud a l'intérêt d'autogérer le stockage (scalabilité). Il peut même se révéler moins cher car il permet une meilleure gestion des ressources (serveurs et mémoire).

Voici l'architecture de notre projet en local :



2. Récupération des données (UiPath)

Le dataset existe chez le site qui possède les données. Cependant, elles sont protégées et ne peuvent être récupérées sous forme d'un dictionnaire. Il est donc nécessaire de le constituer en utilisant des outils de scraping et de récupération automatique de données.

Cela correspond à la brique n°1 du schéma d'architecture. On utilise la technologie UiPath pour automatiser la récupération de données. En effet, les bibliothèques Python comme urllib et request ont été bloquées par LBC (car catégorisées comme des robots). Il a donc fallu mimer le comportement d'un humain : ouverture d'internet explorer, écriture de l'URL de LBC, enregistrement de la page html. Après l'enregistrement de l'URL, le robot ouvre une invite de commande et lance un script Python qui permet d'insérer les données enregistrées dans la base.

Pour note, la page d'accueil qui répertorie les annonces est suffisante puisqu'elle contient toutes les données nécessaires de chaque annonce. Donc pour chaque URL enregistrée, on insère 35 annonces.

3. Insertion des données (Python)

Après récupération des pages via le robot UiPath et enregistrement au format htm, un script python permet de récupérer les données contenues dans le fichier htm et de les insérer dans la base de données mongoDB (en local et dans le cloud).

Cette brique correspond aux briques 1 et 2 dans le schéma d'architecture.

Par exemple, dans l'interface mongoDB, on obtient pour un document (base de données LBC_db1 et collection voitures) :

```
_id: "voitures1867654186"
first_publication_date: "2020-10-27 11:21:08"
expiration_date: "2020-12-26 11:21:08"
index_date: "2020-11-19 07:25:25"
status: "active"
category_id: "2"
category_name: "Voitures"
subject: "RENAULT Grand Scenic Dci 110 EDC Intens, 7 Places + Options, 1ère Main"
body: "Véhicule garanti 12 mois pièces et main d'oeuvre. 1ère Main - 10/10/2..."
ad_type: "offer"
url: "https://www.leboncoin.fr/voitures/1867654186.htm"
> price: Array
> price_calendar: Array
> images: Object
> attributes: Array
> location: Object
> owner: Object
> options: Object
  has_phone: true
  nb_posts: 1
  verification_date: "2020-11-19 07:25:25"
```

Les chevrons « > » indiquent que les attributs contiennent davantage de données (comme options).

4. Analyse des données

a. Requêtes : Sélection

i. Liste des collections de la base de données et nombre de documents

Via python et librairie pymongo

```
import pymongo
import pprint

connex = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
db = connex.LBC_db1

list_collections = db.list_collection_names()
for collection in list_collections:
    print(collection, " : ", db[collection].estimated_document_count())
```



Librairies importées



Connexion à la base de données mongoDB



Affichage du résultat

Le résultat affiché est :

```
vendeur : 6
voitures : 8529
vetements : 208
foire : 60
```

Via Compass (interface graphique de mongoDB) :

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
foire	60	8.1 KB	487.2 KB	1	36.0 KB	
vendeur	6	8.5 KB	50.8 KB	1	36.0 KB	
vetements	208	3.8 KB	790.8 KB	1	36.0 KB	
voitures	8,529	6.2 KB	51.6 MB	1	104.0 KB	

ii. Liste des vendeurs d'une collection (exemple : foire)

Via python et librairie pymongo

```
collection = "foire"

for vendeur in db[collection].distinct("owner.user_id"):
    print(vendeur)
```

Le résultat affiché est :

```
00006007-584a-4425-82eb-2a922bf4be06
000a0988-1ab7-44fb-913f-ffba2d3ddcfd
380d10bf-59ef-4acf-81e9-fc653e242a76
66c57fba-54f2-45dd-9495-895f381d6be7
7207ee7c-8a1a-40b6-b1e3-879a0db5566f
db2f0bc7-d787-40b9-a985-54f566571150
```

Via Compass (interface graphique de mongoDB) :

The screenshot shows the MongoDB Compass interface with the following components:

- Collection:** Labeled "Collection", pointing to the "foire" collection in the left sidebar.
- Requête:** Labeled "Requête", pointing to the aggregation pipeline editor. The pipeline consists of two stages:
 - \$unwind:** A stage to expand arrays. The path is set to "\$owner".
 - \$group:** A stage to group documents by the "owner.user_id" field. The output is a list of documents with the format: `{ "_id": "owner.user_id", "nb": { "$sum": 1 } }`.
- Résultat n°1 :** Labeled "Résultat n°1 : Identifiant du vendeur", pointing to the first result document: `{ "_id": "000a0988-1ab7-44fb-913f-ffba2d3ddcfd", "nb": 10 }`.
- Résultat n°2 :** Labeled "Résultat n°2", pointing to the second result document: `{ "_id": "db2f0bc7-d787-40b9-a985-54f566571150", "nb": 10 }`.
- Nombre de résultats:** Labeled "Nombre de résultats", pointing to the "nb" field in the result documents, which represents the count of announcements for each seller.

b. Requêtes : plus complexes

i. Moyenne des prix des voitures

La requête est écrite en Python :

```
collection = "voitures"
res = db[collection].aggregate([
    {
        "$unwind" : "$price"
    },
    {
        "$group": {
            "_id" : "$_id",
            "price0" : {"$first" : "$price"}
        }
    },
    {
        "$group": {
            "_id": 0,
            "nb": {"$sum": 1},
            "prix_moyen" : {"$avg" : "$price0"}
        }
    }
])
```

Le résultat affiché est :

```
>>> affiche(res)
{'_id': 0, 'nb': 8529, 'prix_moyen': 11320.240942666198}
```

On pourrait nettoyer les données en étudiant plus précisément la distribution de la variable prix pour supprimer les valeurs aberrantes.

En modifiant la requête, on fait apparaître les prix minimum et maximum :

```
>>> affiche(res)
{'_id': 0,
 'nb': 8529,
 'prix max': 780000,
 'prix min': 0,
 'prix_moyen': 11320.240942666198}
```

Il nous apparaît clairement que le prix max (780000€) est aberrant. Il en est de même pour les prix minimums.

ii. Map-Reduce : Liste des marques des voitures

Pour afficher l'ensemble des marques des voitures, on considère la structure des données. En effet, la marque est un attribut, contenu dans l'objet (liste) attributes. Tous les éléments de attributes ont la même structure : ce sont des dictionnaires avec des clés identiques.

```

_id: 1404132324
first_publication_date: "2018-03-21 08:41:47"
index_date: "2020-11-19 06:31:59"
status: "active"
category_id: "2"
category_name: "Voitures"
subject: "Renault Mégane 3 1,5 dci 95 cv,116000 kms"
body: "Renault Mégane 3 1,5 dci 95 cv,116000 kms berline, blanc, 5 cv, 5 port..."
ad_type: "offer"
url: "https://www.leboncoin.fr/voitures/1404132324.htm"
> price: Array
> price_calendar: Array
> images: Object
< attributes: Array
  < 0: Object
    key: "activity_sector"
    value: "1"
    > values: Array
      value_label: "1"
      generic: false
  < 1: Object
    key: "brand"
    value: "Renault"
    > values: Array
      key_label: "Marque"
      value_label: "Renault"
      generic: true
  > 2: Object
  > 3: Object
  > 4: Object
  < 5: Object

```

La marque est donc la valeur en face de la clé « brand » contenue dans attributes.

Pour obtenir la liste des marques, on a donc pris le parti de faire une map-reduce :

```

collection = "voitures"
map = Code("function () { "
  "   for(var i=0;i<this.attributes.length;i++){ "
  "       if(this.attributes[i].key == 'brand') { "
  "           emit(this.attributes[i].value, 1);}"
  "   }"
  "}")
reduce = Code("function (key, values) { "
  "   return values.length;"
  "}")
result = db[collection].map_reduce(map, reduce, "map_reduce_brand")
for res in db.map_reduce_brand.find().sort([("value", pymongo. DESCENDING )]):
    pprint.pprint(res)

```

On obtient le résultat suivant :

{'_id': 'Renault', 'value': 1516.0}	{'_id': 'Fiat', 'value': 217.0}	{'_id': 'Hyundai', 'value': 58.0}
{'_id': 'Peugeot', 'value': 1311.0}	{'_id': 'Nissan', 'value': 190.0}	{'_id': 'Land Rover', 'value': 55.0}
{'_id': 'Citroen', 'value': 871.0}	{'_id': 'Toyota', 'value': 146.0}	{'_id': 'Skoda', 'value': 53.0}
{'_id': 'Volkswagen', 'value': 647.0}	{'_id': 'Seat', 'value': 144.0}	{'_id': 'Volvo', 'value': 47.0}
{'_id': 'Bmw', 'value': 556.0}	{'_id': 'Mini', 'value': 133.0}	{'_id': 'Mazda', 'value': 44.0}
{'_id': 'Mercedes', 'value': 520.0}	{'_id': 'Dacia', 'value': 131.0}	{'_id': 'Suzuki', 'value': 42.0}
{'_id': 'Audi', 'value': 500.0}	{'_id': 'Porsche', 'value': 102.0}	{'_id': 'Chevrolet', 'value': 40.0}
{'_id': 'Ford', 'value': 360.0}	{'_id': 'Alfa Romeo', 'value': 78.0}	{'_id': 'Ds', 'value': 28.0}
{'_id': 'Opel', 'value': 276.0}	{'_id': 'Kia', 'value': 70.0}	{'_id': 'Jeep', 'value': 28.0}

{'_id': 'Mitsubishi', 'value': 27.0}	{'_id': 'Rover', 'value': 6.0}	{'_id': 'Isuzu', 'value': 2.0}
{'_id': 'Jaguar', 'value': 26.0}	{'_id': 'Lancia', 'value': 6.0}	{'_id': 'Alpine-Renault', 'value': 2.0}
{'_id': 'Honda', 'value': 24.0}	{'_id': 'Maserati', 'value': 5.0}	{'_id': 'Lada', 'value': 2.0}
{'_id': 'Lexus', 'value': 22.0}	{'_id': 'Lamborghini', 'value': 5.0}	{'_id': 'Daewoo', 'value': 2.0}
{'_id': 'Smart', 'value': 21.0}	{'_id': 'Austin', 'value': 5.0}	{'_id': 'Simpa JDM', 'value': 1.0}
{'_id': 'Autres', 'value': 19.0}	{'_id': 'Cadillac', 'value': 5.0}	{'_id': 'Casalini', 'value': 1.0}
{'_id': 'Aixam', 'value': 16.0}	{'_id': 'Infiniti', 'value': 5.0}	{'_id': 'Alpine', 'value': 1.0}
{'_id': 'Abarth', 'value': 13.0}	{'_id': 'Ford USA', 'value': 4.0}	{'_id': 'Ferrari', 'value': 1.0}
{'_id': 'Ligier', 'value': 13.0}	{'_id': 'Corvette', 'value': 4.0}	{'_id': 'Mega', 'value': 1.0}
{'_id': 'Dodge', 'value': 11.0}	{'_id': 'Tesla', 'value': 4.0}	{'_id': 'Bertone', 'value': 1.0}
{'_id': 'Saab', 'value': 10.0}	{'_id': 'Buick', 'value': 3.0}	{'_id': 'Chatenet', 'value': 1.0}
{'_id': 'Microcar', 'value': 9.0}	{'_id': 'Daihatsu', 'value': 3.0}	{'_id': 'Cupra', 'value': 1.0}
{'_id': 'Chrysler', 'value': 8.0}	{'_id': 'Aston Martin', 'value': 3.0}	{'_id': 'Iveco', 'value': 1.0}
{'_id': 'Ssangyong', 'value': 8.0}	{'_id': 'Lotus', 'value': 3.0}	{'_id': 'Pontiac', 'value': 1.0}
{'_id': 'Subaru', 'value': 6.0}	{'_id': 'Mg', 'value': 2.0}	{'_id': 'Triumph', 'value': 1.0}

Il y a au total 8478 voitures listées dans ce map reduce, or la collection contient 8529 voitures. Donc il y a 51 voitures qui n'avaient pas de marques répertoriées.

iii. Moyenne des prix des voitures en fonction des marques

Le code est très similaire au précédent (iii). Dans la fonction map, on renvoie le prix de la voiture, puis dans la fonction reduce, on calcule la moyenne des prix :

```
collection = "voitures"
map = Code("function () { "
    "    for(var i=0;i<this.attributes.length;i++){ "
    "        if(this.attributes[i].key == 'brand') { "
    "            emit(this.attributes[i].value, this.price[0]);}"
    "    }"
    "}")
reduce = Code("function (key, values) { "
    "    var total = 0;"
    "    for (var i = 0; i < values.length; i++) { "
    "        total += values[i];"
    "    }"
    "    return total/values.length;"
    "}")
result = db[collection].map_reduce(map, reduce, "map_reduce_price_per_brand")
for res in db.map_reduce_price_per_brand.find().sort([("value", pymongo. DESCENDING )]):
    pprint.pprint(res)
```

On obtient le résultat suivant :

```
{'_id': 'Lamborghini', 'value': 166318.2}
{'_id': 'Ferrari', 'value': 94000.0}
{'_id': 'Maserati', 'value': 69256.0}
{'_id': 'Lotus', 'value': 68800.0}
{'_id': 'Tesla', 'value': 61825.0}
{'_id': 'Aston Martin', 'value': 59266.666666666664}
{'_id': 'Porsche', 'value': 54283.13725490196}
{'_id': 'Cupra', 'value': 50900.0}
{'_id': 'Triumph', 'value': 49910.0}
{'_id': 'Corvette', 'value': 48500.0}
{'_id': 'Dodge', 'value': 37376.36363636364}
{'_id': 'Land Rover', 'value': 25719.272727272728}
{'_id': 'Autres', 'value': 25038.947368421053}
{'_id': 'Jaguar', 'value': 24468.5}
{'_id': 'Ds', 'value': 21506.428571428572}
{'_id': 'Mercedes', 'value': 21395.771153846155}
{'_id': 'Lexus', 'value': 20270.0}
{'_id': 'Audi', 'value': 18331.394}
{'_id': 'Buick', 'value': 17333.666666666668}
{'_id': 'Abarth', 'value': 17220.0}
{'_id': 'Jeep', 'value': 17198.571428571428}
{'_id': 'Infiniti', 'value': 16874.0}
{'_id': 'Volvo', 'value': 16619.468085106382}
{'_id': 'Ford USA', 'value': 16600.0}
{'_id': 'Chevrolet', 'value': 16556.975}
{'_id': 'Subaru', 'value': 16480.0}
{'_id': 'Bmw', 'value': 16004.48561151079}
{'_id': 'Isuzu', 'value': 15345.0}
{'_id': 'Bertone', 'value': 15000.0}
{'_id': 'Honda', 'value': 14662.75}
{'_id': 'Mazda', 'value': 14203.84090909091}
{'_id': 'Casalini', 'value': 13990.0}
{'_id': 'Ford', 'value': 13586.980555555556}
{'_id': 'Cadillac', 'value': 12960.0}
{'_id': 'Mini', 'value': 12311.30827067669}
{'_id': 'Alpine-Renault', 'value': 11500.5}
{'_id': 'Skoda', 'value': 11484.11320754717}
{'_id': 'Kia', 'value': 11323.885714285714}
{'_id': 'Volkswagen', 'value': 11046.911901081916}
{'_id': 'Nissan', 'value': 10655.17894736842}
{'_id': 'Hyundai', 'value': 10321.793103448275}
{'_id': 'Mitsubishi', 'value': 10223.62962962963}
{'_id': 'Austin', 'value': 9700.0}
{'_id': 'Chatenet', 'value': 9490.0}
{'_id': 'Suzuki', 'value': 9160.166666666666}
{'_id': 'Alfa Romeo', 'value': 9142.679487179486}
{'_id': 'Dacia', 'value': 8847.557251908396}
{'_id': 'Toyota', 'value': 8752.787671232876}
{'_id': 'Ligier', 'value': 8368.153846153846}
{'_id': 'Peugeot', 'value': 7592.313501144165}
{'_id': 'Seat', 'value': 7113.888888888889}
{'_id': 'Pontiac', 'value': 7000.0}
{'_id': 'Fiat', 'value': 6834.917050691244}
{'_id': 'Renault', 'value': 6645.701187335092}
{'_id': 'Mg', 'value': 6225.0}
{'_id': 'Citroen', 'value': 6214.107921928818}
{'_id': 'Chrysler', 'value': 6123.75}
{'_id': 'Smart', 'value': 5966.142857142857}
{'_id': 'Opel', 'value': 5950.978260869565}
{'_id': 'Aixam', 'value': 5824.25}
{'_id': 'Microcar', 'value': 5694.222222222223}
{'_id': 'Simpa JDM', 'value': 5500.0}
{'_id': 'Mega', 'value': 5000.0}
{'_id': 'Lancia', 'value': 4151.666666666667}
{'_id': 'Saab', 'value': 3996.6}
{'_id': 'Daewoo', 'value': 3575.0}
{'_id': 'Ssangyong', 'value': 3540.125}
{'_id': 'Daihatsu', 'value': 2699.6666666666665}
{'_id': 'Lada', 'value': 2600.0}
{'_id': 'Iveco', 'value': 2500.0}
{'_id': 'Rover', 'value': 885.0}
{'_id': 'Alpine', 'value': 1.0}
```

On peut enregistrer ce résultat dans un DataFrame (librairie Pandas) puis sauvegarder dans un fichier csv.

Lamborghini : 166318.0	Jaguar : 24468.0	Bmw : 16004.0	Nissan : 10655.0	Simpa JDM : 5500.0
Ferrari : 94000.0	Ds : 21506.0	Isuzu : 15345.0	Hyundai : 10322.0	Mega : 5000.0
Maserati : 69256.0	Mercedes : 21396.0	Bertone : 15000.0	Mitsubishi : 10224.0	Lancia : 4152.0
Lotus : 68800.0	Lexus : 20270.0	Honda : 14663.0	Austin : 9700.0	Saab : 3997.0
Tesla : 61825.0	Audi : 18331.0	Mazda : 14204.0	Chatenet : 9490.0	Daewoo : 3575.0
Aston Martin : 59267.0	Buick : 17334.0	Casalini : 13990.0	Suzuki : 9160.0	Ssangyong : 3540.0
Porsche : 54283.0	Abarth : 17220.0	Ford : 13587.0	Alfa Romeo : 9143.0	Daihatsu : 2700.0
Cupra : 50900.0	Jeep : 17199.0	Cadillac : 12960.0	Dacia : 8848.0	Lada : 2600.0
Triumph : 49910.0	Infiniti : 16874.0	Mini : 12311.0	Toyota : 8753.0	Iveco : 2500.0
Corvette : 48500.0	Volvo : 16619.0	Alpine-Renault : 11500.0	Smart : 5966.0	Rover : 885.0
Dodge : 37376.0	Ford USA : 16600.0	Skoda : 11484.0	Opel : 5951.0	Alpine : 1.0
Land Rover : 25719.0	Chevrolet : 16557.0	Kia : 11324.0	Aixam : 5824.0	
Autres : 25039.0	Subaru : 16480.0	Volkswagen : 11047.0	Microcar : 5694.0	

iv. Moyenne des prix des voitures en fonction de l'âge de la voiture

```
collection = "voitures"
map = Code("function () { "
    "   for(var i=0;i<this.attributes.length;i++){ "
    "       if(this.attributes[i].key == 'regdate') { "
    "           emit(this.attributes[i].value, this.price[0]); } "
    "   } "
    "}")
reduce = Code("function (key, values) { "
    "   var total = 0;"
    "   for (var i = 0; i < values.length; i++) { "
    "       total += values[i]; "
    "   } "
    "   return total/values.length;"
    "}")
result = db[collection].map_reduce(map, reduce, "map_reduce_price_per_year")
affiche(db.map_reduce_price_per_year.find().sort([("_id", pymongo.DESCENDING)]))
df = pd.DataFrame(db.map_reduce_price_per_year.find()).round(0)
df = df.sort_values(by='_id', ascending=False)
```

On obtient la liste suivante :

2020 : 26957.0	2006 : 3948.0	1992 : 1036.0	1967 : 39322.0
2019 : 26180.0	2005 : 3240.0	1991 : 7920.0	1966 : 43077.0
2018 : 23686.0	2004 : 3019.0	1990 : 17569.0	1965 : 27862.0
2017 : 19416.0	2003 : 2974.0	1989 : 9622.0	1964 : 8990.0
2016 : 17228.0	2002 : 3449.0	1988 : 12492.0	1963 : 10.0
2015 : 15353.0	2001 : 2350.0	1987 : 8119.0	1962 : 35015.0
2014 : 13116.0	2000 : 2225.0	1975 : 8083.0	
2013 : 11377.0	1999 : 1792.0	1974 : 34990.0	
2012 : 10828.0	1998 : 1931.0	1973 : 31998.0	
2011 : 7920.0	1997 : 1853.0	1972 : 23230.0	
2010 : 7577.0	1996 : 1723.0	1971 : 25480.0	
2009 : 6495.0	1995 : 31851.0	1970 : 20098.0	
2008 : 5209.0	1994 : 6143.0	1969 : 13862.0	
2007 : 5153.0	1993 : 2506.0	1968 : 21945.0	

v. Nombre de vêtements par catégorie

On s'intéresse maintenant à la collection « vetements » avec 208 lignes de données.

Il y a quatre catégories de vêtements :

- Femme
- Maternité
- Homme
- Enfant

On souhaite connaître la répartition des annonces dans ces catégories.

Voici le code :

```
collection = "vetements"
map = Code("function () {
  for(var i=0;i<this.attributes.length;i++){
    if(this.attributes[i].key == 'clothing_type') {
      emit(this.attributes[i].value_label, this.price[0]);
    }
  }
}")
reduce = Code("function (key, values) {
  return values.length;
}")
result = db[collection].map_reduce(map, reduce, "vetement_categorie_nb")
affiche(db.vetement_categorie_nb.find())
```

On peut visualiser le résultat dans l'interface Compass :

The screenshot shows the MongoDB Compass interface. On the left sidebar, under the 'Collections' tab, the collection 'vetement_categorie_nb' is highlighted. The main panel displays the 'Documents' tab for this collection, showing three documents with their respective values: 'Enfant' (60), 'Femme' (111), and 'Homme' (37). Annotations in French point to the 'Map reduce' operation, the results visualization, and the collections list.

Map reduce

Visualisation des résultats de la map-reduce

Collections : résultats des map-reduce

Il n'y a aucune annonce de vêtements de maternité dans notre base de données. La catégorie avec le plus d'offres sont les vêtements de femme.

vi. Prix des vêtements par catégorie

On s'intéresse aux prix moyens des 4 catégories de vêtements (voir map-reduce précédente).
Voici le code :

```
collection = "vetements"
map = Code("function () { "
    "    for(var i=0;i<this.attributes.length;i++){ "
    "        if(this.attributes[i].key == 'clothing_type') { "
    "            emit(this.attributes[i].value_label, this.price[0]);}"
    "    }"
    "}")
reduce = Code("function (key, values) { "
    "    var total = 0;"
    "    for (var i = 0; i < values.length; i++) { "
    "        total += values[i];"
    "    }"
    "    return total/values.length;"
    "}")
result = db[collection].map_reduce(map, reduce, "vetement_categorie_price")
affiche(db.vetement_categorie_price.find())
```

On obtient le résultat suivant :

```
{ '_id': 'Enfant', 'value': 13.683333333333334 }
{ '_id': 'Femme', 'value': 28.963963963963963 }
{ '_id': 'Homme', 'value': 39.75675675675676 }
```

Les vêtements pour homme sont les plus chers.

En collectant davantage de données (344 données contre 208 précédemment), on obtient ces résultats (via Compass) :

<code>_id: "Enfant"</code> <code>value: 13.214285714285714</code>
<code>_id: "Femme"</code> <code>value: 31.043715846994534</code>
<code>_id: "Homme"</code> <code>value: 32.08064516129032</code>
<code>_id: "Maternité"</code> <code>value: 20</code>

vii. Type de vêtements vendus

De la même façon que les catégories de vêtement, on peut regarder le type de vêtements le plus vendus. La requête s'écrit :

```
map = Code("function () {"  
  " for(var i=0;i<this.attributes.length;i++){ "  
    " if(this.attributes[i].key == 'clothing_category') {"  
      " emit(this.attributes[i].value_label, this.price[0]);}"  
    " }"  
  "}")  
reduce = Code("function (key, values) {"  
  " return values.length;"  
  "}")  
result = db[collection].map_reduce(map, reduce, "vetement_type_nb")  
affiche(db.vetement_type_nb.find().sort([("value",  
pymongo.DESENDING)]))
```

On obtient le résultat suivant :

```
{'_id': 'Manteaux & Vestes', 'value': 62.0}  
{'_id': 'Hauts / T-Shirts / Polos', 'value': 28.0}  
{'_id': 'Pulls / Gilets / Mailles', 'value': 26.0}  
{'_id': 'Robes / jupes', 'value': 25.0}  
{'_id': 'Chemises / Chemisiers', 'value': 11.0}  
{'_id': 'Pantalons', 'value': 11.0}  
{'_id': 'Autre', 'value': 9.0}  
{'_id': 'Jeans', 'value': 7.0}  
{'_id': 'Shorts / Pantacourts / Bermudas', 'value': 6.0}  
{'_id': 'Sous-vêtements & vêtements de nuit', 'value': 5.0}  
{'_id': 'Costumes / Tailleurs', 'value': 4.0}  
{'_id': 'Lingerie', 'value': 4.0}  
{'_id': 'Sports / Danse', 'value': 2.0}  
{'_id': 'Maillots de bain & vêtements de plage', 'value': 1.0}  
{'_id': 'Mariage', 'value': 1.0}
```

Les manteaux et vestes ont le vent en poupe.

Observons maintenant leur prix moyen en modifiant la fonction reduce :

```
reduce = Code("function (key, values) {"  
  " var total = 0;"  
  " for (var i = 0; i < values.length; i++) {"  
    " total += values[i];"  
  " }"  
  " return total/values.length;"  
  "}")  
result = db[collection].map_reduce(map, reduce, "vetement_type_prix")  
affiche(db.vetement_type_prix.find().sort([("value", pymongo.DESENDING)]))
```

Les résultats sont :

```
{'_id': 'Mariage', 'value': 100.0}
{'_id': 'Manteaux & Vestes', 'value': 41.903225806451616}
{'_id': 'Jeans', 'value': 34.857142857142854}
{'_id': 'Pulls / Gilets / Mailles', 'value': 30.653846153846153}
{'_id': 'Lingerie', 'value': 27.0}
{'_id': 'Autre', 'value': 23.44444444444443}
{'_id': 'Costumes / Tailleurs', 'value': 22.5}
{'_id': 'Chemises / Chemisiers', 'value': 20.272727272727273}
{'_id': 'Sports / Danse', 'value': 17.0}
{'_id': 'Robes / jupes', 'value': 15.0}
{'_id': 'Hauts / T-Shirts / Polos', 'value': 13.571428571428571}
{'_id': 'Pantalons', 'value': 11.818181818181818}
{'_id': 'Shorts / Pantacourts / Bermudas', 'value': 8.0}
{'_id': 'Sous-vêtements & vêtements de nuit', 'value': 5.8}
{'_id': 'Maillots de bain & vêtements de plage', 'value': 2.0}
```

a. **Visualisation des données**

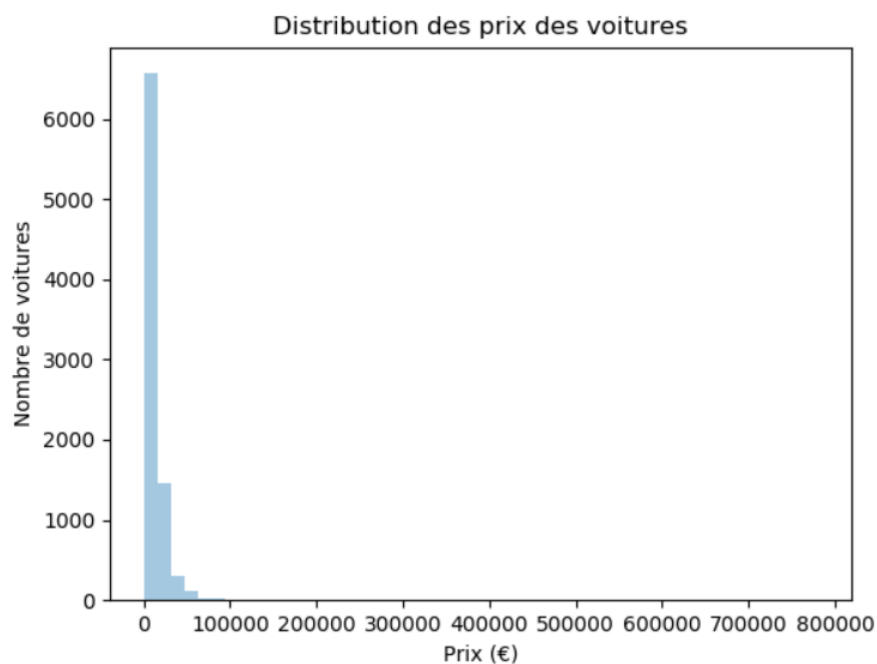
i. Distribution des prix par catégorie

Pour les voitures, nous souhaitons observer la distribution des prix.

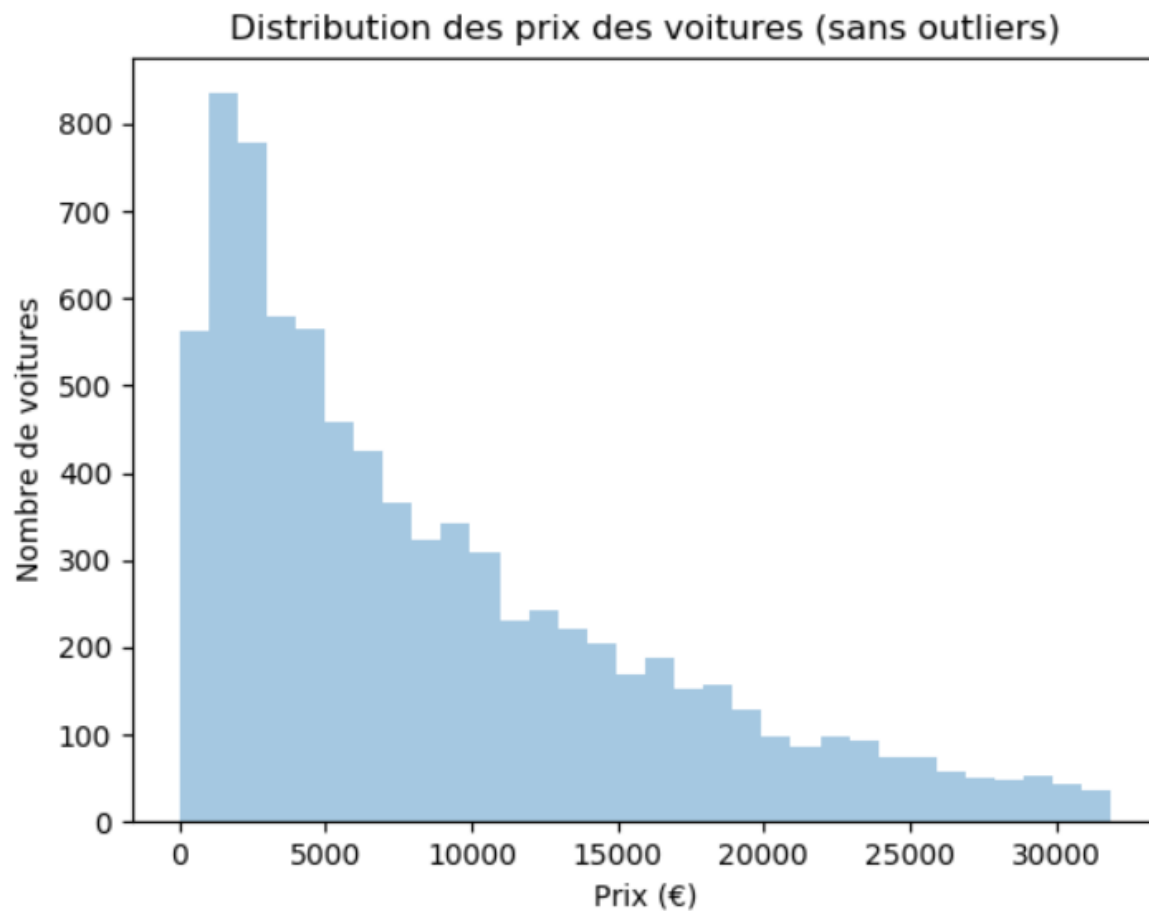
```
import pandas as pd
import seaborn
import matplotlib.pyplot as plt

prix = []
for res in db["voitures"].find():
    prix.append(res["price"][0])

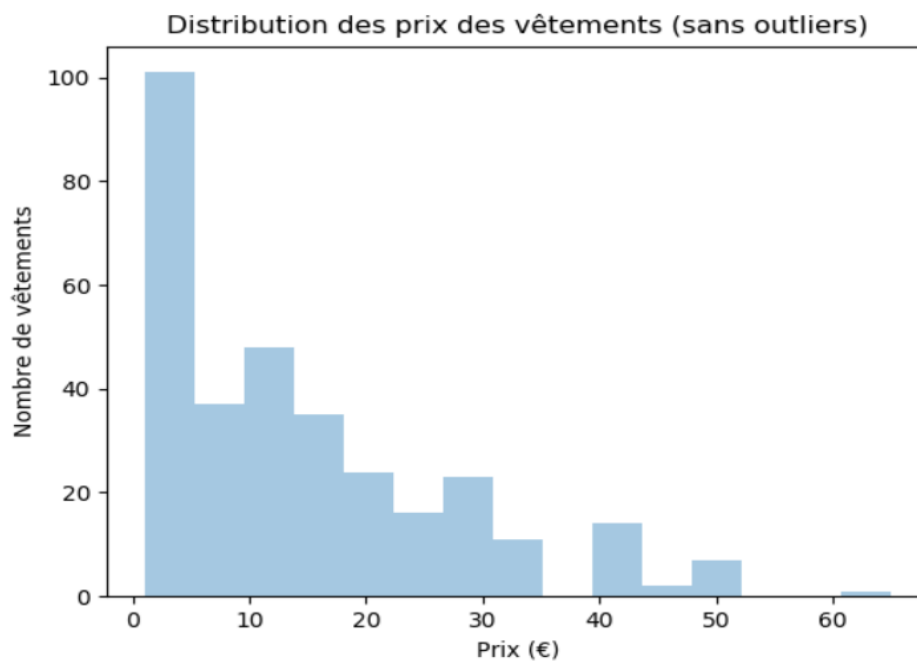
fig = plt.figure(1)
seaborn.distplot(prix, kde=False)
plt.show()
plt.savefig("dist_prix_voiture.png")
```



On remarque que des valeurs aberrantes perturbent la courbe. On va supprimer les outliers. On obtient la distribution suivante :



Pour la distribution des prix des vêtements, on obtient :



Distribution prix par catégorie (countplot)

II. Description des livrables

Pour ce projet, on livre :

- Documentation (rapport) ;
- Captures d'écran et vidéos de démonstration ;
- Code source :
 - o Robot UiPath ;
 - o Scripts Python

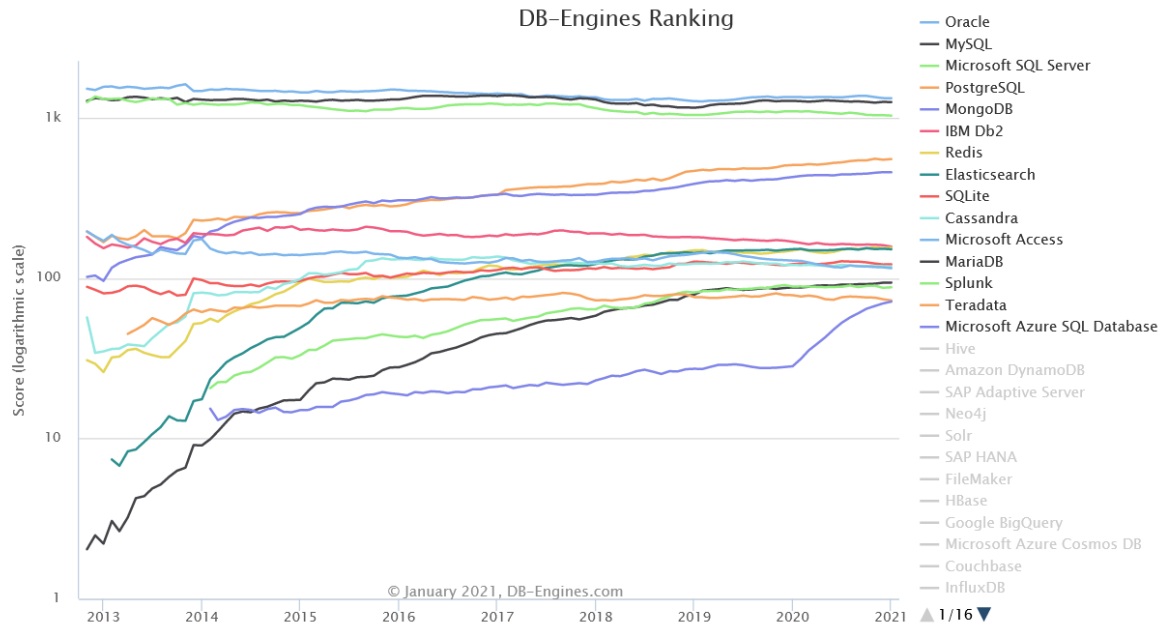
Le code source est organisé comme ici :

- Projet UiPath :
 - o Dossiers associés (.entities, .objects, .screenshots, .settings, .tmh)
 - o Codes UiPath :
 - main_annonces.xaml : code pour le robot qui récupère les pages html du site LBC et lance le script Python insert_mongoDB.py pour insérer les données dans la base de données ;
 - main_vendeurs.xaml : code pour le robot qui récupère les données des pages vendeurs professionnels et les insère dans la collection vendeur
 - verification_annonce.xaml : code pour le robot qui vérifie l'existence des annonces stockées dans la base de données
- Scripts Python :
 - o insert_mongoDB.py : lit une page html et insère les données dans la base de données (local ou cloud) ;
 - o insert_owner_pro_mongoDB.py : lit la page html des vendeurs professionnels et insère les données dans la collection vendeurs.
 - o read_annonces.py : lit les annonces stockées dans notre base de données et crée un fichier csv avec les url et les titres des annonces. Ce script est utilisé pour vérifier que les annonces sont toujours en ligne (avec le robot verification_annonce.xaml)
 - o read_owner.py : similaire au script précédent, il crée un fichier csv qui contient le type de vendeur (professionnel ou particulier) et l'identifiant du vendeur (qui permet de reconstituer l'URL).
 - o requetes.py : script python qui contient toutes les requêtes que nous avons effectuées.
 - o update_map_reduce.py : script Python qui met à jour le résultat des map-reduce dans la base de données.

III. Analyse de la technologie

1. MongoDB : Base de données noSQL documentaire

MongoDB est un leader dans le domaine des bases de données et est classé 5^{ème} en Janvier 2021 dans le classement complet des bases de données. Dans sa catégorie (bases de données noSQL), mongoDB est classé 1^{er}. A fortiori, il est aussi 1^{er} dans la catégorie bases de données noSQL orientées documents.



En effet, dans notre projet, nos données sont assimilées à des documents (format json). Ce format repose sur le format clé-valeur mais avec une extension des champs qui composent le document, comme on peut le voir ici :

```
_id: "voitures1867654186"
first_publication_date: "2020-10-27 11:21:08"
expiration_date: "2020-12-26 11:21:08"
index_date: "2020-11-19 07:25:25"
status: "active"
category_id: "2"
category_name: "Voitures"
subject: "RENAULT Grand Scenic Dci 110 EDC Intens, 7 Places + Options, 1ère Main"
body: "Véhicule garanti 12 mois pièces et main d'oeuvre. 1ère Main - 10/10/2..."
ad_type: "offer"
url: "https://www.leboncoin.fr/voitures/1867654186.htm"
> price: Array
> price_calendar: Array
> images: Object
> attributes: Array
> location: Object
> owner: Object
> options: Object
  has_phone: true
  nb_posts: 1
  verification_date: "2020-11-19 07:25:25"
```

2. ElasticSearch : Base de données noSQL documentaire (moteur de recherche Lucene)

Le site db-engines.com classe Elastic-search dans la catégorie search engines DB. Elastic Search peut aussi contenir des documents type json (comme nos données dans ce projet).

L'intérêt de Elastic Search se situe dans son moteur de recherche Lucene qui permet d'obtenir des scores de similarité lorsqu'il retourne le résultat d'une requête. Pour notre projet, avec des requêtes sur le prix des objets (par exemple), Elastic Search ne s'y prêtait pas.

En revanche, si on avait voulu travailler davantage sur le texte des annonces, Elastic Search était une alternative intéressante.

3. Volume et scalabilité

La scalabilité est une question centrale dans les projets de data science [5]. Ici elle se pose surtout pour les capacités de stockage puisque la récupération de toutes ces données demandera toujours plus de capacité de stockage.

En partant du principe qu'on enregistre environ 3 images pour chaque annonce (environ 50ko par image) et environ 15 champs texte (listés dans la partie 1 Objectif) d'une taille de 100 caractères en moyenne : 1 octet (par caractère) $[6] * 100 * 15 = 1500$ octets soit 1,5 ko. On retient donc 152 ko par annonce. Environ 800 000 nouvelles annonces étaient publiées chaque jour en 2016 [2]. On a donc besoin de $800\,000 * 152\text{ ko} = 1,2\text{ Go}$ par jour. A ce rythme, il faut donc imaginer un système capable d'augmenter régulièrement les capacités de stockage ou un nettoyage/réduction des données stockées. En effet, est-il utile de conserver des données datant de plusieurs années quand les comportements changent très rapidement ?

La deuxième interrogation est la capacité à récupérer toutes ces données. En effet, il faut charger chaque jour, 800 000 pages URL pour récupérer la donnée et charger les anciennes pages URL déjà récupérées pour suivre le temps de durée de vie de l'annonce.

En partant du principe qu'il faut 1 seconde pour charger 1 page internet et récupérer la donnée et qu'il y a 86 400 secondes par jour, il faut déclinier la récupération de données sur plusieurs machines (au moins 10, sûrement 30). Avec 30 machines, 10 permettent de récupérer les données des nouvelles annonces, les 20 autres permettent de mettre à jour la BDD avec le suivi des annonces.

Pour éviter les problèmes de conflit entre différentes annonces, on propose de créer des robots qui récupèrent la donnée selon la catégorie des annonces :

- Véhicules
- Immobilier
- Vacances
- Loisirs
- Animaux et divers
- Mode
- Multimédia
- Services
- Maison
- Matériel Professionnel

Il est possible de plager davantage de machines (avec des robots) pour récupérer la donnée. Les conflits d'écriture dans la BDD seront évités grâce à l'URL unique des annonces.

MongoDB permet de gérer les pannes et la distribution des données, comme toute base de données noSQL.

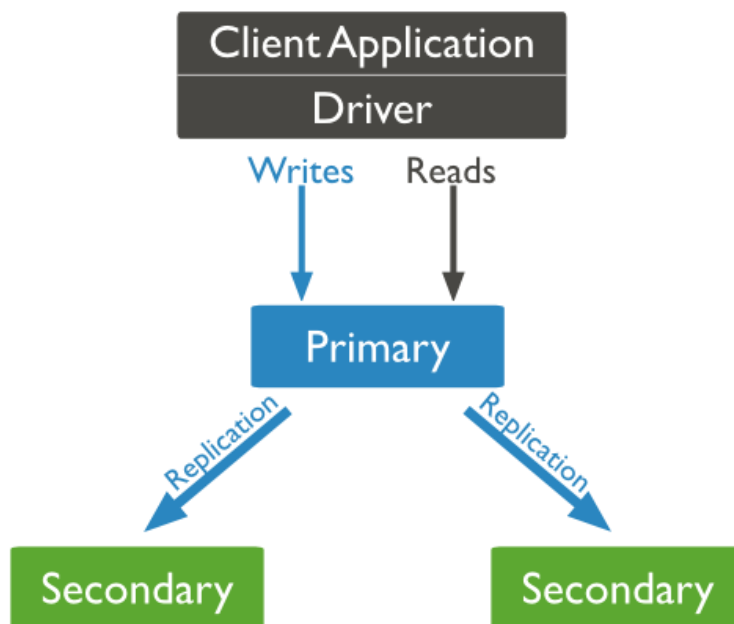
a. Replicaset : Tolérance aux pannes et disponibilité des données

Le ReplicaSet est un outil avec mongoDB qui permet la redondance des données sur plusieurs machines. Cela a deux buts :

- La tolérance aux pannes ;
- La disponibilité des données.

i. Stratégie de déploiement

Un ReplicaSet est composé de 3 serveurs (1 serveur maître et 2 serveurs esclave) comme ci-dessous :



Datacenter 1



Serveur 1 :
Nœud primaire

Serveur 2 :
Nœud secondaire 1

Datacenter 2



Serveur 3 :
Nœud secondaire 2

Pour un ReplicaSet, composé de minimum 3 nœuds, la stratégie de déploiement avec 2 datacenter (dans 2 lieux différents), on placerait chaque nœud sur différentes machines avec au moins un serveur dans chaque datacenter comme représenté au-dessus.

ii. Création d'un ReplicaSet

Instancions un Replica Set :

1. Création d'un répertoire dans le disque C (par exemple) pour chaque serveur :

C:/data/rs1

C:/data/rs2

C:/data/rs3

Le dossier db contient notre base de données initiales (LBC et les collections voitures, vêtements, etc).

2. Dans la console, pour chaque serveur, respectivement :

```
mongod --replSet rs0 --port 27018 --dbpath C:/data/rs1
```

```
mongod --replSet rs0 --port 27019 --dbpath C:/data/rs2
```

```
mongod --replSet rs0 --port 27020 --dbpath C:/data/rs3
```

On obtient le résultat suivant (avec 3 invites de commande) :

The screenshot displays three Notepad++ windows, each showing a different MongoDB log file. The windows are titled 'MINGW64/c/Users/juliette.courigbet'. The logs contain various messages related to MongoDB's internal operations, including database updates, replication status checks, and session management. The logs are formatted with timestamps and structured data fields.

Les serveurs sont lancés.

3. Dans une autre invite de commande, on écrit :

```
mongo --port 27018
```

On obtient le résultat suivant :


```
$ mongo --port 27018
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:27018/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2f4eec15-b626-46c6-99af-d273bdb88544") }
MongoDB server version: 4.4.1
```

En tapant :

rs.initiate();

```
$ mongo --port 27018
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:27018/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2f4eec15-b626-46c6-99af-d273bdb88544") }
MongoDB server version: 4.4.1
rs.initiate ();
{
  "info2" : "no configuration specified. Using a default configuration for the set",
  "me" : "localhost:27018",
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610363572, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1610363572, 1)
}
```

4. On ajoute les deux serveurs esclaves au ReplicaSet :

rs.add("localhost:27019");

rs.add("localhost:27020");

```
rs.add("localhost:27019");
{
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610364676, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1610364676, 1)
}
rs.add("localhost:27020");
{
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610364696, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1610364696, 1)
}
```

Définissons l'arbitre :

1. Il faut créer un dossier arb avec le chemin d'accès suivant (par exemple) : C:/data/arb
2. Dans une autre invite de commande :

mongod --port 30000 --dbpath C:/data/arb --replSet rs0

3. Dans l'invite de commande ayant lancé les serveurs :

```
rs.addArb("localhost:30000");
{
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610370530, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1610370530, 1)
}
```

On obtient le statut du ReplicaSet en tapant rs.status();

```
rs.status();
{
  "set" : "RS1",
  "date" : ISODate("2021-01-14T08:16:41.346Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "majorityVoteCount" : 3,
  "writeMajorityCount" : 3,
  "votingMembersCount" : 5,
  "writableVotingMembersCount" : 4,
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    },
    "lastCommittedWallTime" : ISODate("2021-01-14T08:16:31.627Z"),
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityWallTime" : ISODate("2021-01-14T08:16:31.627Z"),
    "appliedOpTime" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1610612191, 1),

```

On obtient la liste des membres du ReplicaSet et leur fonction (PRIMARY, SECONDARY, ARBITER).

```

"members" : [
  {
    "_id" : 0,
    "name" : "localhost:27018",
    "health" : 1,
    "state" : 1,
    "stateStr" : "PRIMARY",
    "uptime" : 593,
    "optime" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    },
    "optimeDate" : ISODate("2021-01-14T08:16:31Z"),
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "electionTime" : Timestamp(1610611841, 2),
    "electionDate" : ISODate("2021-01-14T08:10:41Z"),
    "configVersion" : 5,
    "configTerm" : 1,
    "self" : true,
    "lastHeartbeatMessage" : ""
  },
  {
    "_id" : 1,
    "name" : "localhost:27019",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 206,
    "optime" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    },
    "optimeDurable" : {
      "ts" : Timestamp(1610612191, 1),
      "t" : NumberLong(1)
    }
  }
]

```

Ainsi, ici, le serveur sur le port d'écoute 27018 est le serveur primaire (maître).

On importe nos données sur le serveur primaire (27018) :

```
mongoimport --db LBC --collection voitures --port 27018 C:/data/voitures.json --jsonArray
```

On crée donc une base de données LBC et la collection voitures

iii. Résistance aux pannes

Pour vérifier la résistance aux pannes, on va fermer la console du serveur primaire (port 27018) :

```

S :352}}
{"t":{"$date":"2021-01-14T09:27:02.955+01:00"},"s":"I", "c":"STORAGE", "id":2281, "ctx":"consoleTerminate","msg":"shutdown: removing fs lock..."}
{"t":{"$date":"2021-01-14T09:27:02.960+01:00"},"s":"I", "c":"-", "id":4784931, "ctx":"consoleTerminate","msg":"Dropping the scope cache for shutdown"}
{"t":{"$date":"2021-01-14T09:27:02.960+01:00"},"s":"I", "c":"CONTROL", "id":20565, "ctx":"consoleTerminate","msg":"Now exiting"}
{"t":{"$date":"2021-01-14T09:27:02.960+01:00"},"s":"I", "c":"CONTROL", "id":23138, "ctx":"consoleTerminate","msg":"Shutting down","attr":{"exitCode":12}}

```

Un nouveau serveur est désigné comme le serveur primaire. Pour cela, on fait `rs.status()` ;

C'est le serveur 27019 qui est désigné comme le serveur primaire.

On a bien accès à la base de données sur mongoDBCompass (port 27019) :

MongoDB Compass - localhost:27019/LBC

Connect View Help

Local

4 DBS 8 COLLECTIONS

☆ FAVORITE

HOST
localhost:27019

CLUSTER
Primary

EDITION
MongoDB 4.4.1 Community

Filter your data

LBC

voitures

admin

config

local

CREATE COLLECTION

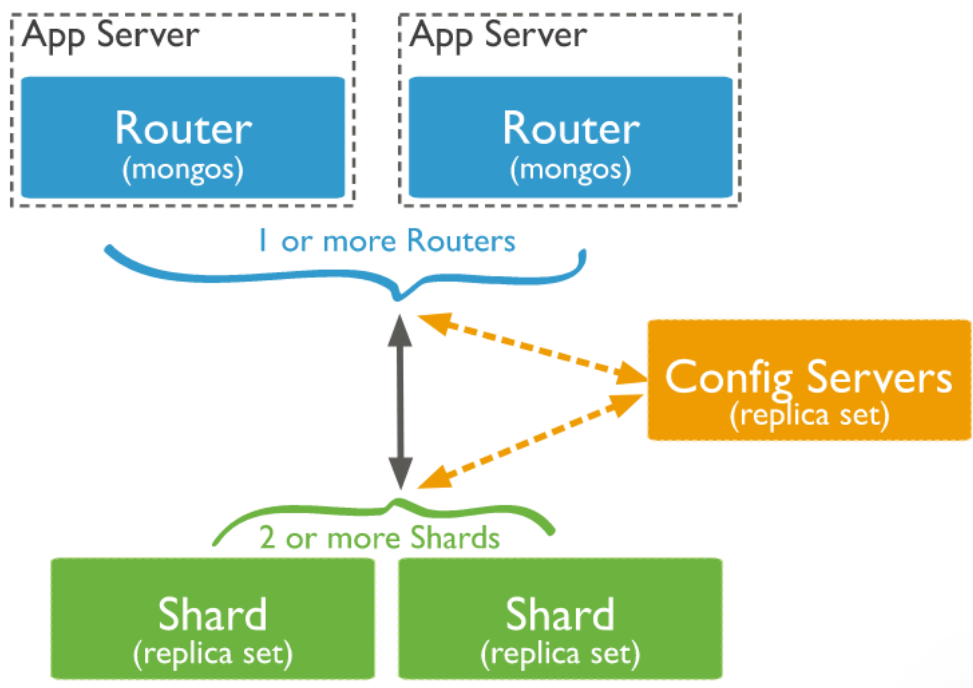
Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
voitures	8,529	6.2 KB	51.6 MB	1	104.0 KB	

b. Sharding : Scalabilité horizontale

i. Prérequis

1. Avec la version 4.4 de mongoDB il faut installer les tools <https://docs.mongodb.com/database-tools/installation/installation-windows/>
On obtient donc dans le répertoire C:/Program Files/MongoDB, le dossier Server et Tools.
2. Ajouter les chemins
C:\Program Files\MongoDB\Server\4.4\bin
C:\Program Files\MongoDB\Tools\100\bin
à la variable d'environnement Path

ii. Structure du cluster

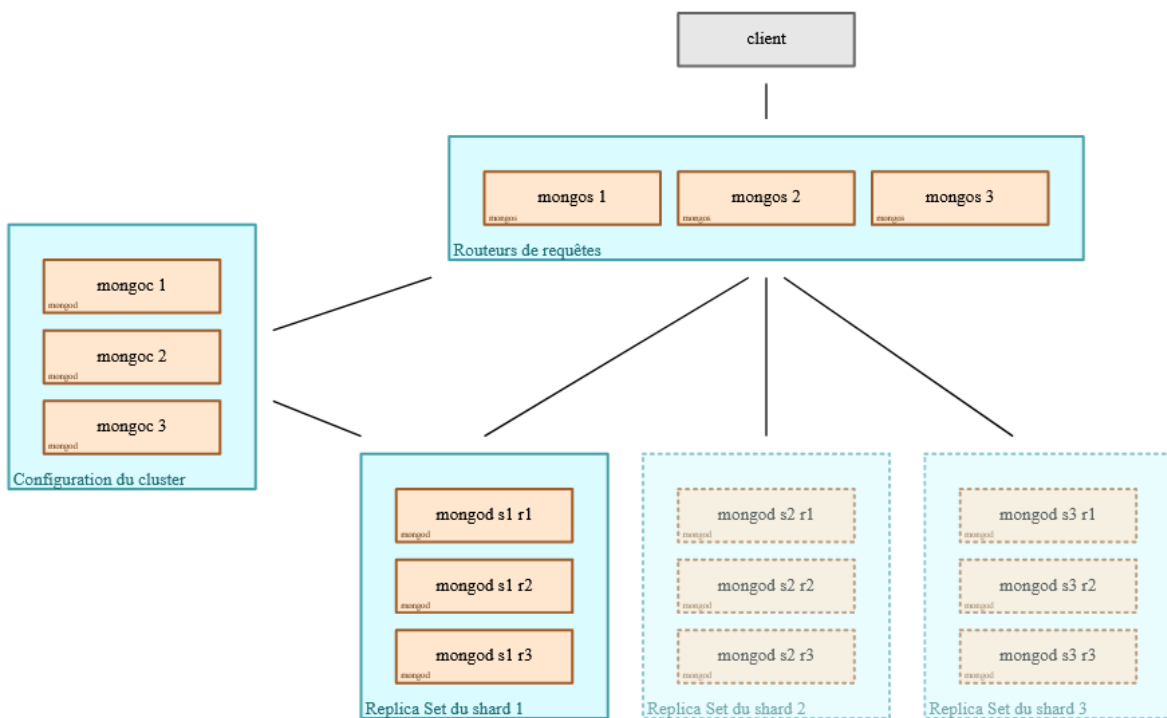


Un cluster est composé au minimum de :

- Un router (mongos) : il s'occupe du routage des requêtes. Pour gérer la tolérance aux pannes, on peut créer 2 routers.
- Un config server : c'est un serveur et il est configuré en ReplicaSet. Il contient la connaissance du réseau (routers et clusters). Ils gèrent les informations de répartitions de charges sur les différents shards et la structure des ReplicaSet.
- Deux shards au minimum : ils sont configurés en ReplicaSet (3 serveurs avec redondance des données). Ils contiennent l'ensemble des données (chunks). Ils peuvent contenir plusieurs chunks.

Pour notre démonstration, nous avons la structure suivante :

- Un router ;
- Un configServer ;
- Trois shards.



iii. Réalisation du cluster

Il faut tout d'abord créer les dossiers associés à ces 4 éléments (configServer et shards) dans le répertoire C:/data :

C:\data		Rechercher dans : data		
	Nom	Modifié le	Type	Taille
je	arb	12/01/2021 14:11	Dossier de fichiers	
	configdb	12/01/2021 16:10	Dossier de fichiers	
le	db	12/01/2021 14:11	Dossier de fichiers	
me	rs1	12/01/2021 14:11	Dossier de fichiers	
its	rs2	12/01/2021 14:11	Dossier de fichiers	
	rs3	12/01/2021 14:11	Dossier de fichiers	
	sh1	12/01/2021 16:09	Dossier de fichiers	
	sh2	12/01/2021 16:10	Dossier de fichiers	
	sh3	12/01/2021 16:10	Dossier de fichiers	
	voitures.json	12/01/2021 15:40	JSON File	65 471 Ko

On crée les dossiers sh1, sh2, sh3 et configdb.

1. Lancement du serveur configServer en ReplicaSet.

Dans une fenêtre de commande, on lance :

```
mongod --configsvr --replSet configReplSet --port 28000 --dbpath C:/data/configdb
```

Dans une autre fenêtre, on se connecte au port 28000 (donc au configServer) et on l'initie :

```
juliette.courgibet@FR-L4648312 MINGW64 ~
$ mongo --port 28000
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:28000/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("a2cd2e83-0d7b-4a3f-97de-622f56ef141a") }
MongoDB server version: 4.4.1
rs.initiate()
{
  "info2" : "no configuration specified. Using a default configuration for the set",
  "me" : "localhost:28000",
  "ok" : 1,
  "$gleStats" : {
    "lastOpTime" : Timestamp(1610461018, 1),
    "electionId" : ObjectId("000000000000000000000000")
  },
  "lastCommittedOpTime" : Timestamp(0, 0),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461018, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1610461018, 1)
}
```

Avec la commande : `mongo --port 28000`

Puis `rs.initiate()`;

2. Lancement des serveurs de données (shards) en ReplicaSet.

Dans 3 fenêtres de commande, on lance :

```
mongod --shardsvr --replSet sh1 --port 27031 --dbpath C:/data/sh1
mongod --shardsvr --replSet sh2 --port 27032 --dbpath C:/data/sh2
mongod --shardsvr --replSet sh3 --port 27033 --dbpath C:/data/sh3
```

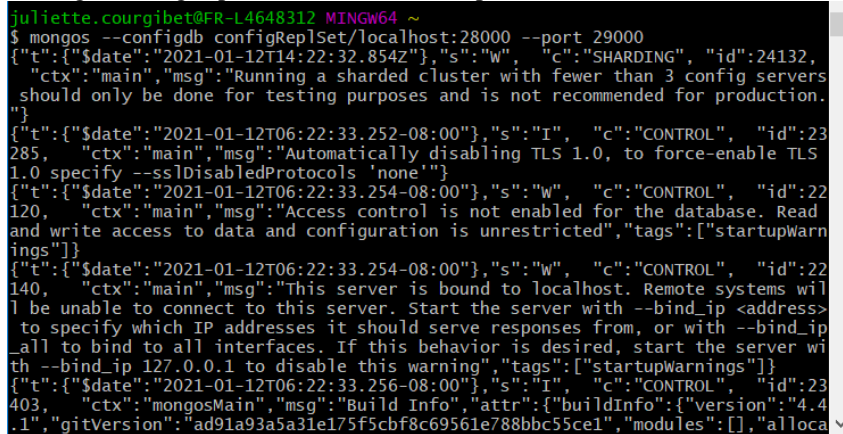
Puis dans une autre invite de commande, on initie les shards :

```
mongo --port 27031 --eval "rs.initiate()"
mongo --port 27032 --eval "rs.initiate()"
mongo --port 27033 --eval "rs.initiate()"
```

3. Lancement du routeur et connexion aux shards

Pour lancer le routeur, dans une invite de commande, on tape pour le connecter au configServer du port 28000 :

```
mongos --configdb configReplSet/localhost:28000 --port 29000
```



```
juliette.courgibet@FR-L4648312 MINGW64 ~  
$ mongos --configdb configReplSet/localhost:28000 --port 29000  
{ "t": { "$date": "2021-01-12T14:22:32.854Z" }, "s": "w", "c": "SHARDING", "id": 24132, "ctx": "main", "msg": "Running a sharded cluster with fewer than 3 config servers should only be done for testing purposes and is not recommended for production." }  
{ "t": { "$date": "2021-01-12T06:22:33.252-08:00" }, "s": "i", "c": "CONTROL", "id": 23285, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'" }  
{ "t": { "$date": "2021-01-12T06:22:33.254-08:00" }, "s": "w", "c": "CONTROL", "id": 22120, "ctx": "main", "msg": "Access control is not enabled for the database. Read and write access to data and configuration is unrestricted", "tags": [ "startupWarnings" ] }  
{ "t": { "$date": "2021-01-12T06:22:33.254-08:00" }, "s": "w", "c": "CONTROL", "id": 22140, "ctx": "main", "msg": "This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning", "tags": [ "startupWarnings" ] }  
{ "t": { "$date": "2021-01-12T06:22:33.256-08:00" }, "s": "i", "c": "CONTROL", "id": 23403, "ctx": "mongosMain", "msg": "Build Info", "attr": { "buildInfo": { "version": "4.4.1", "gitVersion": "ad91a93a5a31e175f5cbf8c69561e788bbc55ce1", "modules": [], "allLoca
```

On se connecte ensuite au port 29000 (routeur) pour ajouter les shards :

Dans une autre invite de commande, on tape :

```
mongo --port 29000  
sh.addShard("sh1/localhost:27031");  
sh.addShard("sh2/localhost:27032");  
sh.addShard("sh3/localhost:27033");
```

Dans la figure suivante, on voit le résultat obtenu (et attendu) :



```
juliette.courgibet@FR-L4648312 MINGW64 ~  
$ mongo --port 29000  
MongoDB shell version v4.4.1  
connecting to: mongod://127.0.0.1:29000/?compressors=disabled&gssapiServiceName=mongod  
Implicit session: session { "id" : UUID("22ffaa26-d0fa-4ccb-bfe7-eab333662228") }  
MongoDB server version: 4.4.1  
sh.addShard("sh1/localhost:27031");  
{  
  "shardAdded" : "sh1",  
  "ok" : 1,  
  "operationTime" : Timestamp(1610461422, 6),  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1610461422, 6),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),  
      "keyId" : NumberLong(0)  
    }  
  }  
}  
sh.addShard("sh2/localhost:27032");  
{  
  "shardAdded" : "sh2",  
  "ok" : 1,  
  "operationTime" : Timestamp(1610461441, 5),  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1610461441, 5),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),  
      "keyId" : NumberLong(0)  
    }  
  }  
}  
sh.addShard("sh3/localhost:27033");  
{  
  "shardAdded" : "sh3",  
  "ok" : 1,  
  "operationTime" : Timestamp(1610461453, 5),  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1610461453, 5),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),  
      "keyId" : NumberLong(0)  
    }  
  }  
}
```

iv. Ingestion de données dans le cluster

Pour cette étape, on crée une base de données et une collection voitures. Pour cela, on récupère les données issues de notre base (localhost :27017) en enregistrant un fichier au format.json (exportation de données).

Dans la même invite de commande connectée au port 29000, on crée une base de données LBC en autorisant le sharding et en créant une collection voitures :

```
use LBC;
switched to db LBC
sh.enableSharding("voitures");
{
  "ok" : 1,
  "operationTime" : Timestamp(1610461491, 7),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461491, 7),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
sh.enableSharding("LBC");
{
  "ok" : 1,
  "operationTime" : Timestamp(1610461510, 6),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461510, 6),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
db.createCollection("voitures");
{
  "ok" : 1,
  "operationTime" : Timestamp(1610461526, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461526, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
db.voitures.createIndex({"_id":1});
{
  "raw" : {
    "sh3/localhost:27033" : {
      "numIndexesBefore" : 1,
      "numIndexesAfter" : 1,
      "note" : "all indexes already exist",
      "ok" : 1
    }
  },
  "ok" : 1,
  "operationTime" : Timestamp(1610461543, 38),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461543, 38),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
sh.shardCollection("LBC.voitures",{"_id":1});
{
  "collectionsharded" : "LBC.voitures",
  "collectionUUID" : UUID("d9f5a0dd-4a88-40d7-a7d5-a394f0f047b8"),
  "ok" : 1,
  "operationTime" : Timestamp(1610461575, 18),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610461575, 19),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

On importe maintenant les données dans une autre invite de commande avec :


```
mongoimport --db LBC --collection voitures --port 29000 C:/data/voitures.json --jsonArray
```

```
$ mongoimport --db LBC --collection voitures --port 29000 C:/data/voitures.json
--jsonArray
2021-01-12T15:44:11.183+0100    connected to: mongodb://localhost:29000/
2021-01-12T15:44:13.409+0100    8529 document(s) imported successfully. 0 document(s) failed to import.
```

On obtient le statut du cluster :

```
$ mongo --port 29000 --eval "sh.status()"
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:29000/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("3b09ffe9-8250-4024-8c35-4bcdcb6e0d8c") }
MongoDB server version: 4.4.1
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5ffdaf5b514ea7183c35faa6")
  }
  shards:
    { "_id" : "sh1", "host" : "sh1/localhost:27031", "state" : 1 }
    { "_id" : "sh2", "host" : "sh2/localhost:27032", "state" : 1 }
    { "_id" : "sh3", "host" : "sh3/localhost:27033", "state" : 1 }
  active mongoses:
    "4.4.1" : 1
  autosplit:
    Currently enabled: yes
  balancer:
    Currently enabled: yes
```

```
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5ffdaf5b514ea7183c35faa6")
  }
  shards:
    { "_id" : "sh1", "host" : "sh1/localhost:27031", "state" : 1 }
    { "_id" : "sh2", "host" : "sh2/localhost:27032", "state" : 1 }
    { "_id" : "sh3", "host" : "sh3/localhost:27033", "state" : 1 }
  active mongoses:
    "4.4.1" : 1
  autosplit:
    Currently enabled: yes
  balancer:
    Currently enabled: yes
    Currently running: no
    Failed balancer rounds in last 5 attempts: 0
    Migration Results for the last 24 hours:
      682 : Success
  databases:
    { "_id" : "LBC", "primary" : "sh3", "partitioned" : true, "version" : { "uuid" :
    UUID("6e7cb2b0-b620-454e-a33b-fc45a10933de"), "lastMod" : 1 } }
```

```

LBC.voitures
  shard key: { "_id" : 1 }
  unique: false
  balancing: true
  chunks:
    sh3    1
    { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : sh3
Timestamp(1, 0)
  { "_id" : "config", "primary" : "config", "partitioned" : true }
  config.system.sessions
    shard key: { "_id" : 1 }
    unique: false
    balancing: true
    chunks:
      sh1    342
      sh2    341
      sh3    341
      too many chunks to print, use verbose if you want to force print
    { "_id" : "voitures", "primary" : "sh3", "partitioned" : true, "version" : { "uuid" :
UUID("6fa5223d-126a-4103-9a85-74391a07aad1"), "lastMod" : 1 } }

```

Conclusion

Tous les objectifs initiaux ont été remplis :

- Récupération des données du LBC (annonces) ;
- Insertion des données dans une base de données adaptée à nos données ;
- Mise à jour des données automatisée ;
- Requête sur notre base de données ;
- Stockage en local et dans le cloud ;
- Recherche et implémentation des outils de distribution des données pour le stockage local.

Tutoriels et documentation

- [1] <https://solwos.com/site-de-petites-annonces-leboncoin-base-de-donnees-protegee/>
- [2] <https://www.base-de-donnees.com/base-de-donnees-lbc/>
- [3] <https://medium.com/databook/des-donn%C3%A9es-%C3%A0-la-cr%C3%A9ation-de-valeur-a5a3f344c845>
- [4] <https://comarketing-news.fr/la-valeur-reelle-de-nos-donnees-personnelles/>
- [5] <https://itsocial.fr/enjeux-it/enjeux-infrastructure/datacenter/quest-scalabilite-5-meilleurs-articles-autour-de-scalabilite/>
- [6] <https://pageperso.lis-lab.fr/stefano.facchini/python/slides8.pdf>

<https://www.programmevitam.fr/ressources/Doc0.26.0/html/archi/archi-exploit-infra/services/mongodb.html>

http://chewbii.com/wp-content/uploads/2015/11/TP_mongodb.pdf

<https://openclassrooms.com/en/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4474611-protégez-vous-des-pannes-avec-les-replicaset>