[page de garde]

Sommaire

[1. Dossier d’Architecture Technique 4](#_Toc71213190)

[1.1. Représentation fonctionnelle 4](#_Toc71213191)

[1.2. Représentation applicative 4](#_Toc71213192)

[1.3. Représentation technique / architecture 5](#_Toc71213193)

[1.3.1. Solution finale 5](#_Toc71213194)

[1.3.2. Autre solution : logstash 6](#_Toc71213195)

[1.3.3. Evolution de la solution : python / flask 7](#_Toc71213196)

[2. Spécification fonctionnelle générale 9](#_Toc71213197)

[3. Cahier de recette 11](#_Toc71213198)

[4. Manuel 12](#_Toc71213199)

[4.1. Installation 12](#_Toc71213200)

[4.1.1. Pré-requis 12](#_Toc71213201)

[4.1.2. Serveur elastic 12](#_Toc71213202)

[4.1.3. Dashboard kibana 13](#_Toc71213203)

[4.2. Utilisation 15](#_Toc71213204)

[5. Démo 16](#_Toc71213205)

# Dossier d’Architecture Technique

## Sujet

Le projet *“traffic rennes”* a pour objet la création d’un outil de visualisation permettant de visualiser en temps réel le trafic routier de l’agglomération de Rennes. Ceci dans l’objectif de permettre aux chauffeurs de la société “Transport Rennes” (spécialisé dans le logistique) de pouvoir adapter leurs trajets ou prévenir à l’avance leurs clients d’un potentiel retard.

## Représentation fonctionnelle

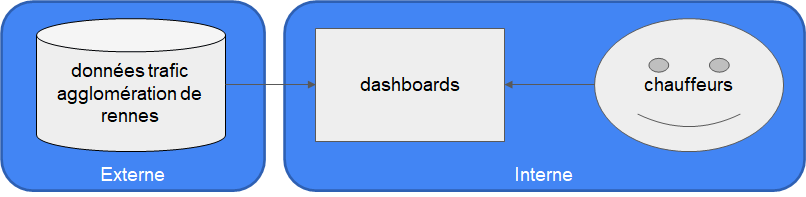


Schéma : représentation fonctionnelle de la demande

## Représentation applicative

Externe :

Source de données : API rennesmetropole > etat du trafic en temps réel

* Lien : <https://data.rennesmetropole.fr/explore/dataset/etat-du-trafic-en-temps-reel/information/>
* Limites :
  + données mise à jour toutes les 3 minutes
  + 1000 exportations par appel à l’api
  + niveau de confiance >=50% à appliquer

Intermédiaire (Externe ←→ Interne) :

Transfert des données depuis la partie “externe” (api) vers la partie “interne” (elk).

Interne :

Données et dashboards : suite ELK

* Stockage des données en interne : elasticsearch
* Réalisation des dashboards : kibana

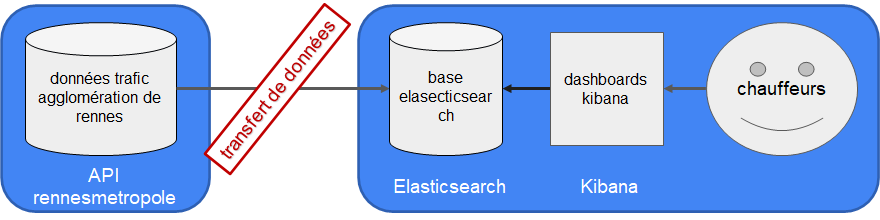


Schéma : représentation applicative de la demande

## Représentation technique / architecture

### Solution finale

A la suite de la solution logstash, un script python de bas niveau pour fournir un terrain d'entente avec elasticsearch nous paraît un avantage. Le script python est léger et la libraire elasticsearch de python nous offre la possibilité de paramétrer la configuration de manière aussi diverses que logstash. En plus des possibilités illimitées de python notamment dans la transformation de données.

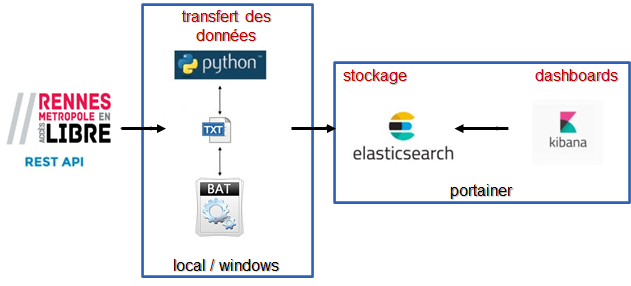


Schéma : représentation de l’architecture technique de la solution.

En premier lieu, la création d’un programme python qui va réaliser le transfert des données. Le transfert comprend :

1. l’appel à l’api et la récupération des données en json
2. le nettoyage et transformations de ces données (détails dans partie 2. spécification..)
3. l’envoie des données vers elasticsearch

Ces 3 étapes constituent le socle de la solution. Elles sont réalisées en boucles (autant de fois que nécessaires) créant ainsi un flux-continue de mise-à-jour des données en temps réel dans elasticsearch.

Ensuite, nous utilisons un fichier batch permettant d 'exécuter le programme python. L’avantage de cette pratique est que l’on peut via le planificateur de tâche de windows, planifier son lancement automatique à une heure précise et ce de manière répétée. En plus de cela, elle offre la possibilité de pouvoir afficher des messages de suivi (log) directement dans le terminal windows.

De plus, pour permettre au client de pouvoir définir ses propres paramètres de configuration, un fichier texte (fichier des paramètres) est à disposition. Dans celui-ci on peut y définir les valeurs des 6 principaux paramètres requis par le programme python.

Ces paramètres doivent être définis sous la forme "nom = valeur" (attention aux espaces), et permettra au client de définir :

|  |  |  |  |
| --- | --- | --- | --- |
| paramètre | définition | type | par défaut |
| index\_name | nom de l'index | string | traffic\_rennes |
| index\_init | créer ou mettre-à-jour l’index | boolean | False |
| traffic\_nb\_rows | nombre de ligne par requête api | integer | 1000 |
| traffic\_reliability | niveau de confiance des données (en %) | integer | 50 |
| traffic\_time\_interval | durée d'attente entre chaque flux (en s) | integer / eval (\*) | 60\*3 |
| traffic\_time\_max | durée total du flux-continue (en s) | integer / eval (\*) | 60\*60\*2 |

Note : pour traffic\_time\_interval et traffic\_time\_max, on a la possibilité de définir leurs valeurs sous forme d’une opération numérique, ce qui permet la lecture lorsque l’on souhaite définir une durée assez élevée en secondes. Par exemple, *.*

Note : Un autre programme python (de test), permet au client de tester et contrôler les pré-requis nécessaires au programme python (de transfert des données) (cf. chapitre 3. cahier de recette).

### Autre solution : logstash

“Logstash” est l’outil de la suite ELK permettant d’importer des données dans elasticsearch depuis des sources externes (csv, odbc, api, ...). Il dispose de nombreux plugins dont “*HTTP Poller*” qui permet de réaliser l’opération dans le cas où la source externe est une api.

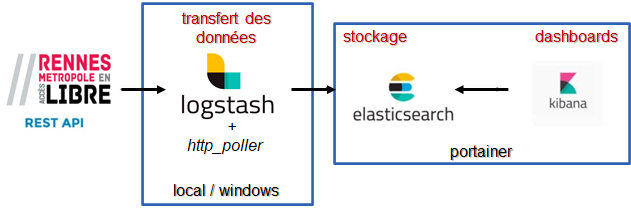


Schéma : représentation de l’architecture logstash

Avec un fichier de configuration pour démarrer Logstash et créer un pipeline, on peut définir les paramètres tels que les entrées, les filtres, les sorties, les conditions et l'intervalle (temps en secondes) de récupération des statistiques.

Le mappage des données ne nous a pas donné un résultat prévisionnel.

### Evolution de la solution : python / flask

“Flask” est un micro - framework python permettant de créer des applications web avec python. l’avantage est qu’Il est très léger et permet de garder un noyau simple et propose de nombreuses extensions pour ajouter des fonctionnalités.

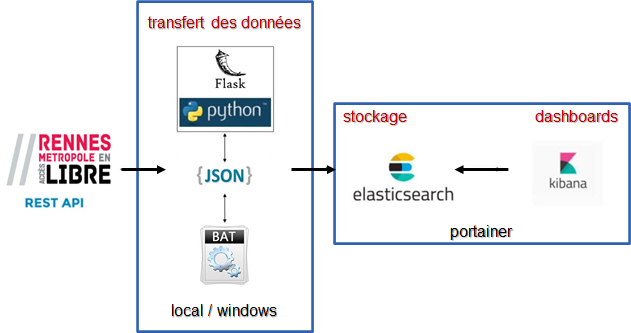


Schéma : architecture de la solution avec python et flask

Il nous permet de construire un formulaire plus intuitif pour sauvegarder les paramètres de configuration que d’éditer le fichier texte *($1.3.3 Solution finale)* . Les données sont sauvegardées au format json .

l’avantage de JSON est qu ‘il est utilisé pour sérialiser et transmettre des données structurées.

On avait la possibilité de construire une page web exécutant la même fonctionnalité que le batch mais le processus de traitement de données prenait plus de temps avant d’afficher une page web . L’architecture restant est identique par rapport à la solution finale

### 

# Spécification fonctionnelle générale

A faire ici : Une spécification fonctionnelle générale, expliquant comment les données de l’API seront récoltées, les transformations que la donnée subira avant d’être stockée dans une base de données.

Pour rappel notre solution comporte 2 grandes étapes que sont :

* transfert des données
* réalisation de dashboards

Les données seront récoltées via une requête http (get) sous format json puis transférer à elasticsearch.

Par défaut, elasticsearch n’arrive pas à déclarer automatiquement les champs géospatiaux au format adéquat. De ce fait, Kibana ne les reconnaît pas aussi, et donc impossible de réaliser de cartes.

Pour y remédier, on doit réaliser le mapping (spécification du format d’un champ) de ces champs là “manuellement” lors de l’étape de transfert des données.

L’étape transfert des données comprend 3 sous-étapes :

1. l’appel à l’api et la récupération des données en json
2. le nettoyage et transformations de ces données
   * transformation des données (\*)
3. l’envoie des données vers elasticsearch
   * connection à elasticsearch
   * la définition du mapping adéquat (\*\*)
   * création d’un index vide sous elasticsearch (\*\*\*)
   * export des données vers elasticsearch

Lors de la sous-étape de transformation des données (\*), on réalise :

* premier filtrage pour ne garder que le contenu du champ “”records” qui contient les données de trafics
* filtrage des données lorsque *“traveltimereliability >= 50”*

La sous-étape création de l’index vide (\*\*\*) est optionnelle. Elle n’est utile que si l’utilisateur souhaite créer un nouvel index. Dans le cas contraire, l’index sera mis-à-jour avec les nouvelles données importées.

La définition du mapping (\*\*) ne concerne que les champs géospatiales, à savoir :

* fields.geo\_point\_2d (positions), type “geo\_point”
* fields.geo\_shape (portion routes), type “geo\_shape”
* geometry.coordinates (positions), type “geo\_point”

Note : contrairement aux champs *geo\_point\_2d* et *geo\_shape*, le champ geometry ne dispose que d’un sous-champs “coordinates” (et non “type” et “coordinates”), d’où la nécessité déclarer directement ce sous-champs en tant que geo\_point.

# Cahier de recette

A faire ici : Un cahier de recette expliquant l’ensemble des tests que vous avez réalisés pour vous assurer de la bonne qualité des données.

Liste des tests effectués :

|  |  |
| --- | --- |
| test effectué avec succès | commentaire |
| requête get de l’api retourne 200 comme réponse | si l’api ne répond pas, on retourne un json vide |
| ping sur le serveur elastic retourne True | si le serveur ne répond pas on arrête le programme |
| trafficrennes\_transfertdata\_utils.py existe ? | script python avec les fonctions |
| trafficrennes\_transfertdata\_parameters.txt existe ? | fichier des paramètres |
| index\_name est str ? | paramètre nom de l’index |
| index\_init vaut True/False ? | paramètre créer ou maj l’index |
| traffic\_nb\_rows est int ? | paramètre nombre de lignes à importer par requête |
| traffic\_reliability est int ? | paramètre niveau de confiance |
| traffic\_time\_interval est int ? | paramètre durée d’attente entre chaque flux |
| traffic\_time\_max est int ? | paramètre durée total du flux continue |
| traffic\_time\_max >= traffic\_time\_interval ? |  |

# Manuel

A faire ici : Un manuel d’installation de la solution (pour que l’on puisse déployer l’outil sur notre infrastructure).

## Installation

### Pré-requis

* docker
  + portainer

modifier le pare-feu local de Windows pour créer une adresse de

loopback.

* + elasticsearch

Installez la version JDK (minimum la version 8) prise en charge par

elasticsearch.

* + kibana

Pas de configuration minimum requise.

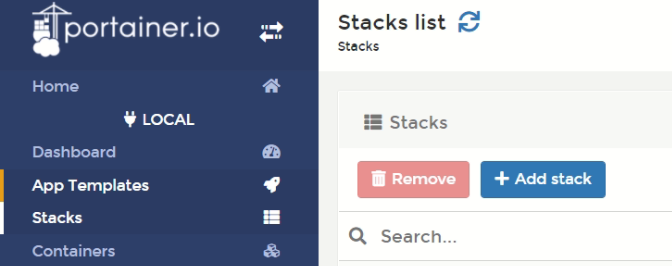
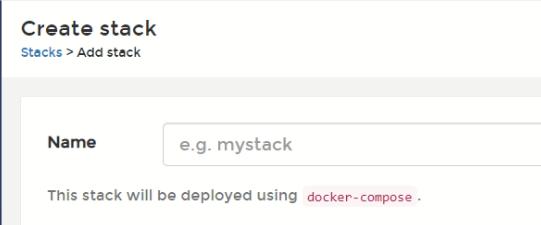
* windows
  + python v3.9.0
* Programme disponible sur git, et comprend :
  + programme bat à lancer (.bat)
  + programme python (.py)
  + script python avec les fonctions perso (.py)
  + fichier des paramètres (.txt)
  + script docker-compose (.yml)
  + données de création du dashboard (.ndjson)

### Serveur elastic

Création des containers elasticsearch et kibana (portainer):

Ouvrir docker > portainer (port 9000) > stacks > add stack > docker-compose\* > deploy

Le contenu de la stack (\*) correspond au contenu du docker-compose (à copier-coller).

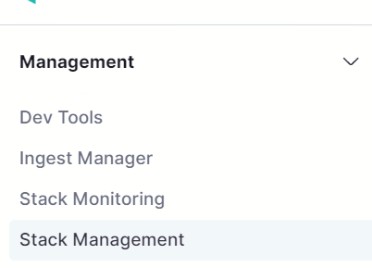
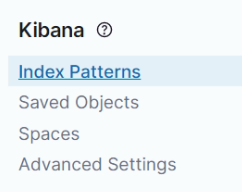
### Dashboard kibana

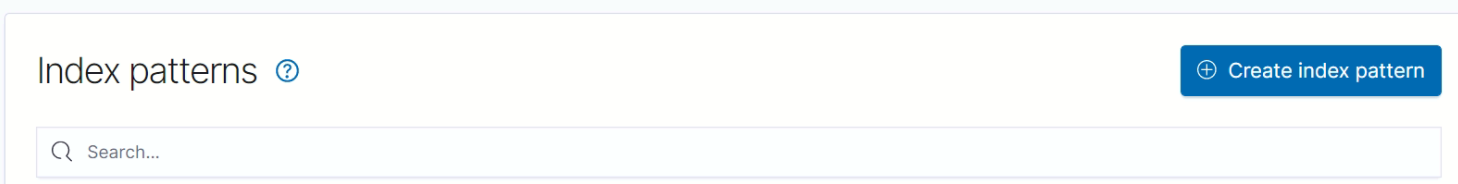
Pré-requis : serveur elasticsearch et kibana actifs, et un **index créer et non vide**.

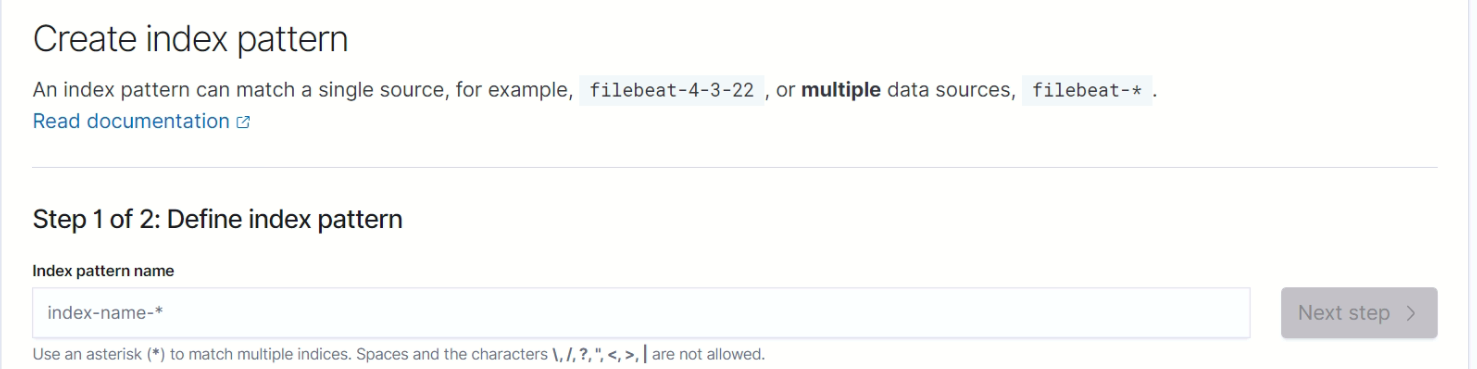
Créer un index pattern (kibana):

Utile pour que kibana puisse reconnaître le nouvel index créer dans elasticsearch.

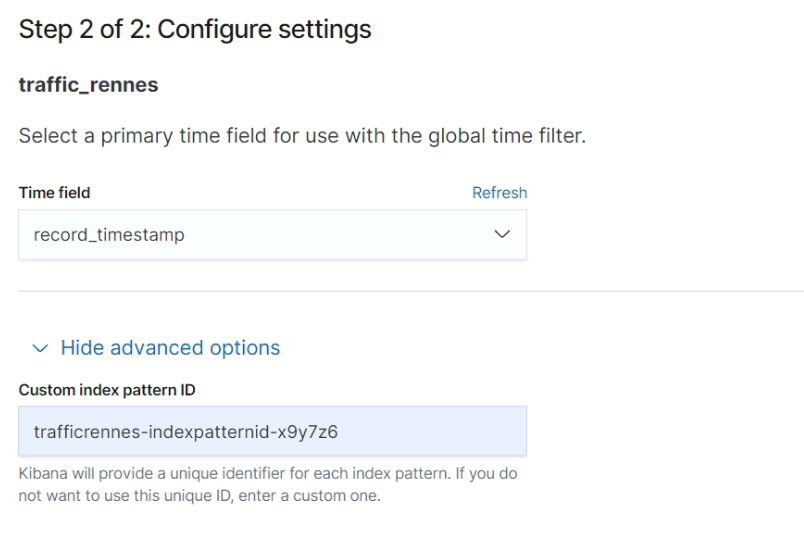
Menu management > stack management : kibana > index patterns :

* create index pattern  
  
* step 1 of 2 :
  + définissez le nom voulu pour que sa match que le nom de l’index créer
  + next step

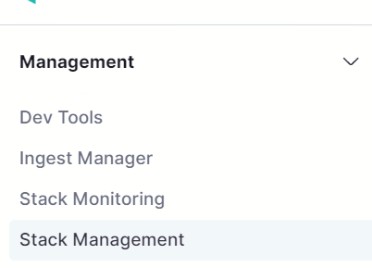
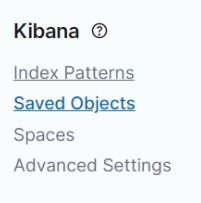


* step 2 of 2 :
  + time field : “fields.datetime”, il s’agit ici du champ de filtre temporel
  + show advanced options > custom index pattern ID : “trafficrennes-indexpatternid-x9y7z6”, on utilise le même id que celui importé
  + create index pattern

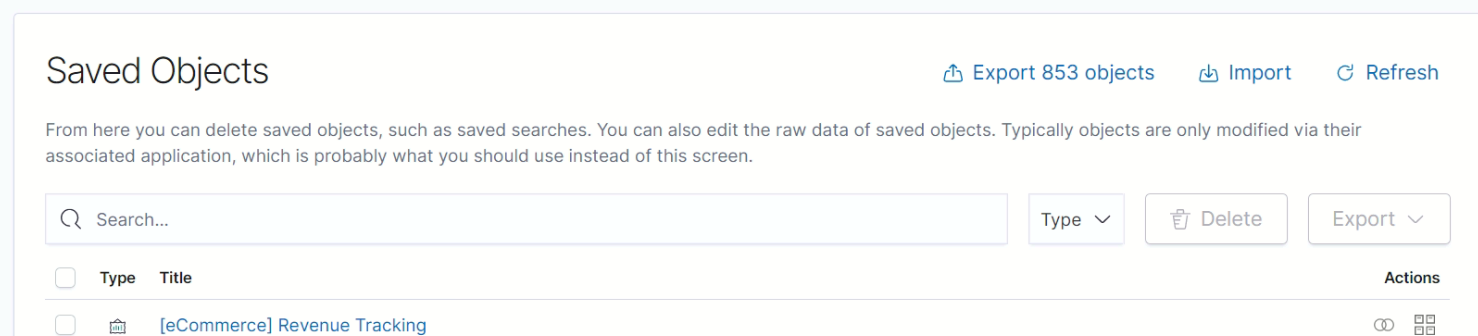


Charger le dahboard et ses composantes (kibana):

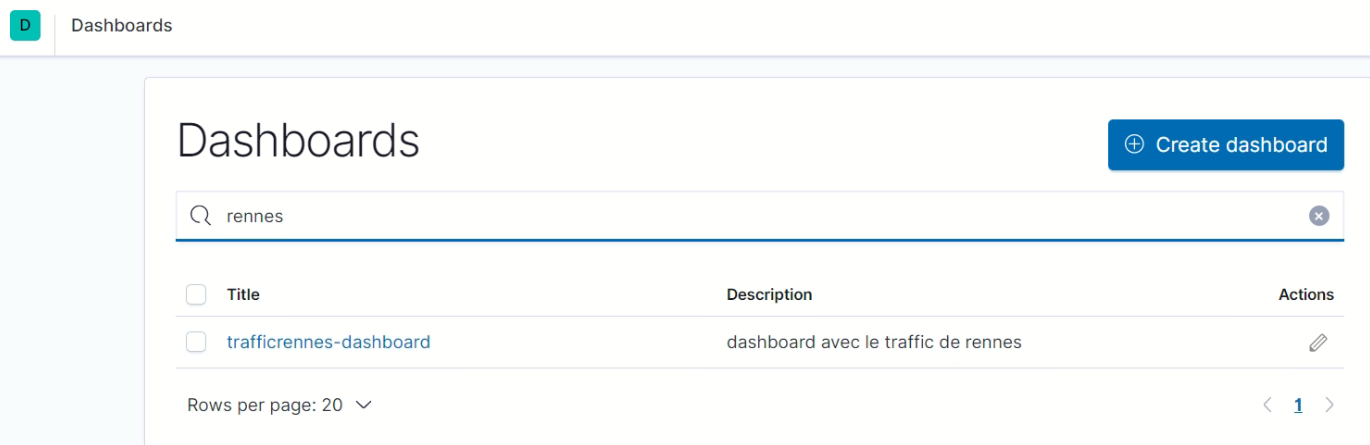
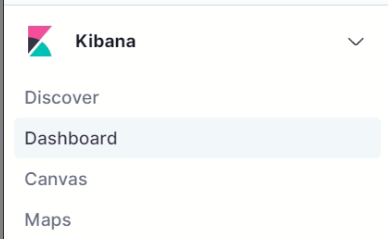
Menu management > stack management : kibana > saved objects :

* import : sélectionner le fichier “trafficrennes\_dashboard.ndjson”



* le dashboard porte le nom “trafficrennes-dashboard”, dispo depuis le Menu *kibana > dashboard*.



## Utilisation

Pré-requis : serveur elasticsearch et kibana actifs.

Par défaut :

Les fichiers

Personnalisé :

# Démo

