

Création d'un outil de visualisation du trafic en temps réel
de l'agglomération de Rennes.

Projet traffic rennes

Auteurs:

Fred DUARTE, Khaly NIANG, Zaina TOIBIBOU

Sommaire

1	Sujet.....	3
2	Dossier d'Architecture Technique	3
2.1	Représentation fonctionnelle	3
2.2	Représentation applicative	3
2.3	Représentation technique / architecture.....	4
2.3.1	Solution finale : python	4
2.3.1.1	Le transfert des données	5
2.3.1.2	La réalisation de dashboards.....	5
2.3.2	Autre solution : logstash	6
2.3.3	Evolution de la solution : python / flask	7
3	Spécification fonctionnelle générale	8
4	Cahier de recette	9
5	Manuel.....	10
5.1	Installation	10
5.1.1	Prérequis.....	10
5.1.2	Serveur elastic	10
5.1.3	Dashboard kibana.....	11
5.2	Utilisation.....	13
5.2.1	Avec la configuration par défaut.....	13
5.2.2	Avec une configuration personnalisée	13
5.2.3	La log	13
6	Démo	14

1 Sujet

Le projet “*traffic rennes*” a pour objectif 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.

2 Dossier d'Architecture Technique

Dans le DAT on y présente le choix des solutions technologiques, avec motivations et justifications ainsi que le schéma de l’architecture technique retenu.

2.1 Représentation fonctionnelle

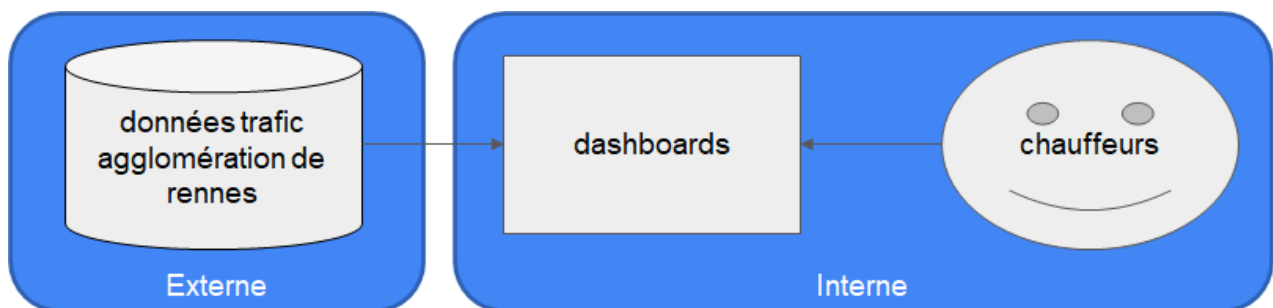


Schéma 1 – représentation fonctionnelle de la demande

On a ici la représentation fonctionnelle de la demande. En effet, on souhaite réaliser un(des) dashboard(s) avec le trafic en temps réel de rennes sur lequel pourront s’appuyer les chauffeurs afin d’adapter leurs trajets. Mais avant cela, il faudra bien trouver la source externe qui fournira les données adéquates.

2.2 Représentation applicative

Externe :

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

- Lien : <https://data.rennesmetropole.fr/explore/dataset/etat-du-traffic-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

- elasticsearch : sgbd nosql orienté document
- kibana : interface utilisateur et réalisation de visualisations (dashboards, ..)

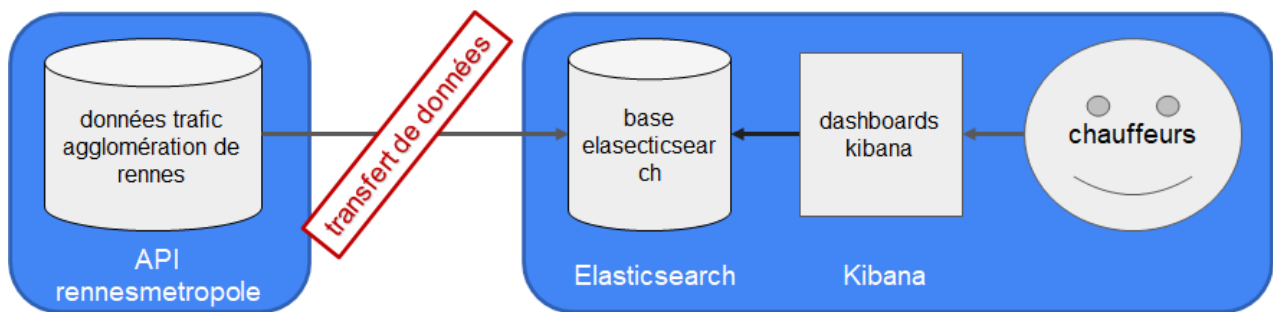


Schéma 2 – représentation applicative de la solution

Ayant plus d'information sur les outils à utiliser, on définit ici la représentation applicative de la future solution. On sait sur quelle source de données s'appuyer, et sur quels outils en interne on a à disposition pour le transfert de ces données depuis la source externe vers la base en interne.

2.3 Représentation technique / architecture

2.3.1 Solution finale : python

Après la non-validation de la solution logstash (cf. 2.3.2-Autre solution : logstash), l'utilisation de python pour fournir un terrain d'entente avec elasticsearch nous paraît plus avantageuse. L'utilisation de python est légère et sa librairie « elasticsearch » donne la possibilité de se connecter à elasticsearch et d'y importer des données tout comme logstash. En plus des possibilités illimitées qu'offre python (notamment dans la transformation de données), celui-ci est bien plus simple en utilisation que logstash.

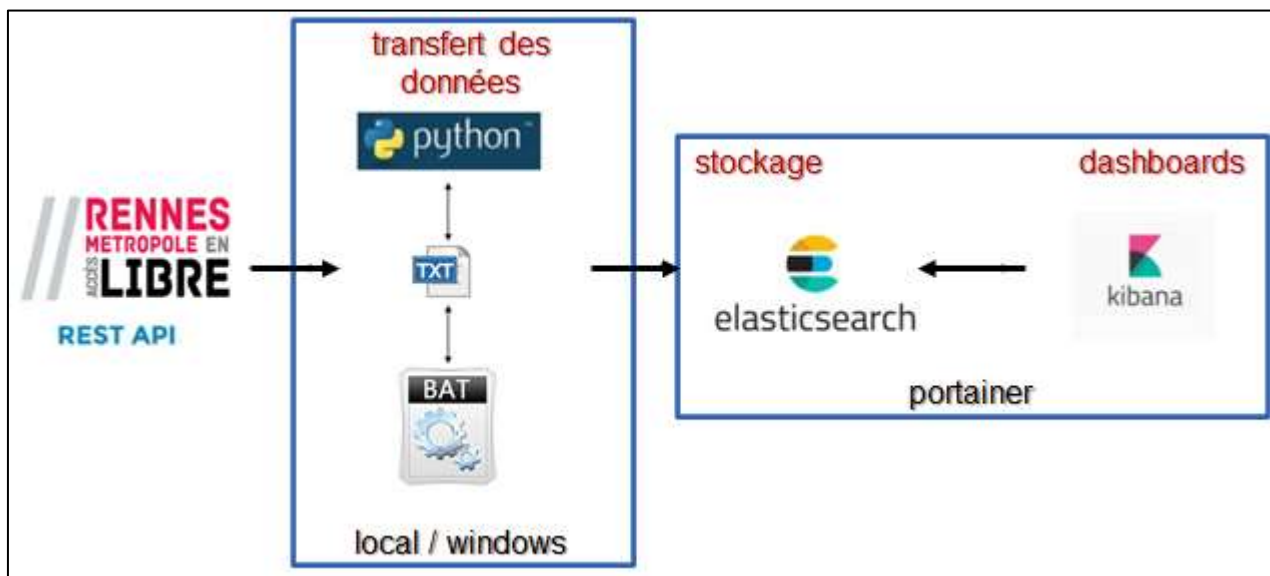


Schéma 3 – représentation de l'architecture technique de la solution

Ci-dessous l'architecture finale de notre solution, avec notamment l'utilisation de python pour la réalisation du transfert de données entre l'api et elasticsearch. Ensuite kibana viendra s'alimenter depuis elasticsearch et ainsi permettre la création de dashboards dynamiques.

2.3.1.1 Le transfert des données

Et donc, on a en premier lieu, un programme python qui va réaliser l'étape de transfert des données. Cette étape comprend les sous-étapes suivantes :

1. l'appel à l'api et la récupération des données en json
2. le nettoyage de ces données (plus de détails dans 3-Spécification fonctionnelle générale)
3. l'injection de ces données dans elasticsearch

Ces 3 sous-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

Tableau 1 – liste des paramètres

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, $60 * 3 \Rightarrow (60s = 1min) * 3 \Rightarrow 3min$.

Note : Un autre programme python (de test), permet au client de tester et contrôler les prérequis nécessaires au programme python (de transfert des données) (cf. Cahier de recette).

Le transfert des données a pour finalité la création (ou mis-à-jour) d'un index¹ elasticsearch.

2.3.1.2 La réalisation de dashboards

Tout comme elasticsearch, kibana (spécialisé dans la visualisation) fait partie de la suite ELK. Les 2 sont donc hyper-compatibles, et peuvent facilement être utilisés ensemble. En effet, les dashboards réalisés sous kibana vont venir s'alimenter en données depuis l'index elasticsearch. Les données de l'index elasticsearch étant mises-à-jour en temps réel (via le flux-continu) garantira en même temps le côté « temps réel » des résultats fournis par les dashboards kibana (le trafic rennais).

¹ Index elasticsearch : représente une collection de document de la même famille, c'est l'équivalent d'une table sql.

2.3.2 Autre solution : logstash

Logstash est l'outil de la suite ELK permettant l'injection de données dans elasticsearch et ce 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.

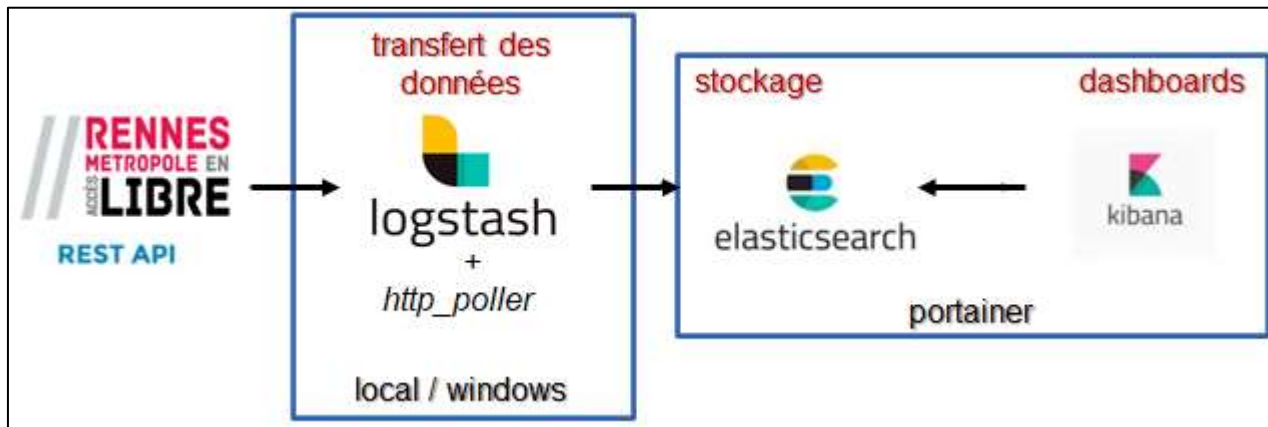


Schéma 4 – représentation de l'architecture logstash

Pour l'utilisation de logstash, on passe par la réalisation d'un script de configuration (au format .conf). Dans celui-ci on peut, sous forme de pipeline, y définir différentes opérations tels que les entrées, les filtres, les sorties, les conditions et l'intervalle (temps en secondes) de récupération des statistiques. Il peut donc être utilisé en tant qu'ETL.

On a décidé d'abandonner cette solution car le mapping automatique des données (lors de la création d'un index) ne nous a pas donné de résultat utilisable (champs géospatiales non reconnus, cf. 3-Spécification fonctionnelle générale pour plus de détails).

2.3.3 Evolution de la solution : python / flask

Flask est un micro-framework python permettant de créer des applications web avec python. L'avantage de celui-ci est qu'il est très léger et permet de garder un noyau simple et propose de nombreuses extensions pour ajouter des fonctionnalités.

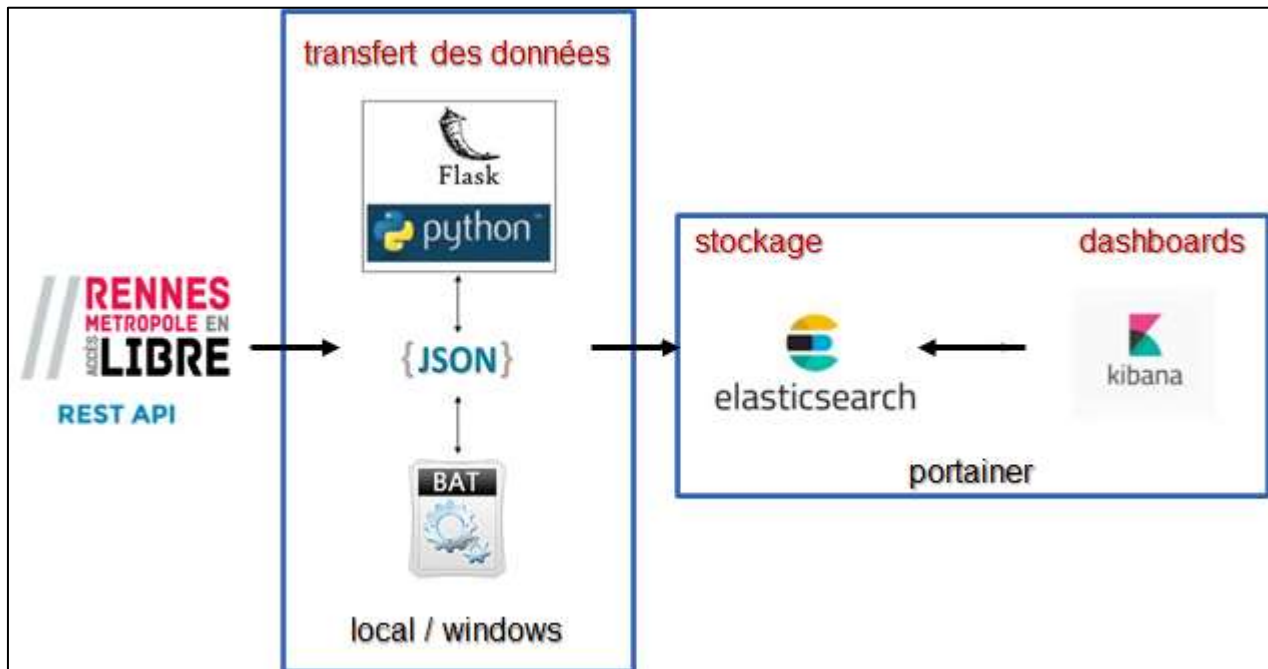


Schéma 5 – architecture de la solution avec python et flask

Il nous permet de construire un formulaire plus intuitif pour sauvegarder les paramètres de configuration (plutôt que d'éditer le fichier texte, cf. 2.3.1-Solution finale : python). Les données sont alors sauvegardées au format json. L'avantage du 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, son fonctionnement nécessitait la fin du programme python pour afficher la log, ce qui est difficilement acceptable.

3 Spécification fonctionnelle générale

On explique ici comment les données de l'API sont récoltées, ainsi que les transformations qu'elles subiront avant d'être stockée dans une base de données.

Pour rappel notre solution comporte 2 grandes étapes que sont :

- Le transfert des données
- La réalisation de dashboards

Les données seront récoltées via une requête http (get) sous format json puis injectées 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 la création de l'index, ce pourquoi on décide de le faire 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 de ces données
 - transformation des données (*)
3. l'injection de ces données dans elasticsearch
 - connection à elasticsearch
 - 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 :

- filtrage de 1^{er} niveau pour ne garder que le contenu du champ « **records** »
 - ce champ contient les données de trafics
- filtrage de 2nd niveau pour ne garder que les données répondant au critère
 - « *traveltimereliability* >= 50 » : pour la fiabilité des données
 - (voir 2.2-Représentation applicative)

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 du sous-champ « coordinates » (pas de champ « type »), d'où la nécessité déclarer directement le sous-champs en tant que « geo_point ».

4 Cahier de recette

Le cahier de recette explique l'ensemble des tests réalisés nous permettant d'assurer le bon fonctionnement du programme.

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 elasticsearch 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 ?	paramètres de configuration (détails voir – liste des paramètres)
index_init vaut True/False ?	
traffic_nb_rows est int ?	
traffic_reliability est int ?	
traffic_time_interval est int ?	
traffic_time_max est int ?	
traffic_time_max >= traffic_time_interval ?	

Tableau 2 – liste des tests effectués

Ces tests concernent la v3.2 de la solution. Ils sont réalisés via un programme python (accompagné d'un script batch de lancement), de noms « unittest-v3.py » et « unittest-v3.bat ».

5 Manuel

Il s'agit ici du manuel d'installation et d'utilisation de la solution.

5.1 Installation

5.1.1 Prérequis

Docker et 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.

Python : v3.9.0 (ou version équivalent, python3 de manière générale)

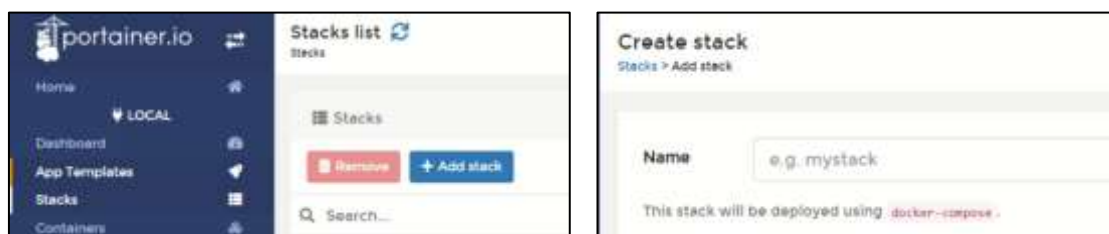
Programme disponible sur git, et comprend :

- programme batch (.bat) : lance le programme python ci-dessous
- programme python (.py) : réalise le transfert des données
- script python utils (.py) : contient les fonctions créées par les développeurs
- fichier des paramètres (.txt) : pour les paramètres de configuration
- script docker-compose (.yml) : pour la création d'une stack sous portainer
- fichier de sauvegarde du dashboard (.ndjson) : permet d'importer le dashboard kibana

5.1.2 Serveur elastic

Création d'une stack avec les containers elasticsearch et kibana (depuis portainer) :

- ouvrir docker > portainer (port 9000) > stacks > add stack > docker-compose* > deploy
- le contenu de la stack (*) correspond au contenu du script docker-compose (à copier-coller)



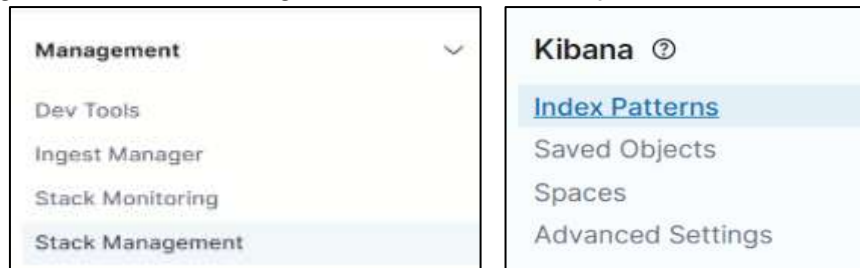
Screen 1 – créer une stack sur portainer

5.1.3 Dashboard kibana

Prérequis : serveurs elasticsearch (port 9200) et kibana (port 5601) actifs, et un **index créé et non vide** (avec le bon mapping). Pour cela on peut lancer le programme python (via le .bat) pour qu'il crée l'index avec le bon mapping.

Créer un index-pattern **avec un ID personnalisé** : nécessaire pour que kibana puisse reconnaître le nouvel index créé dans elasticsearch. A réaliser depuis l'interface utilisateur de kibana :

- Menu management > stack management : kibana > index patterns :



Screen 2 – accéder au menu index-patterns

- create index pattern



Screen 3 – créer un index-pattern (1/3)

- step 1 of 2 :
 - définissez le nom voulu pour que sa match que le nom de l'index créer
 - next step



Screen 4 – créer un index-pattern (2/3)

- step 2 of 2 : très important
 - time field : on défini ici du champ de filtre temporel utilisé pour la mise-à-jour des dashboards
 - en utilisant « fields.datetime », on va réaliser nos dashboards en se basant sur la date de mise à jour du trafic,
 - alors qu'avec « record._timestamp », on utilise plutôt la date de l'écriture des données dans l'index elasticsearch,
 - il y a une différence de temps qui peut être plus ou moins importante, on conseil d'utiliser « fields.datetime » pour avoir des représentatifs de la réalité.
 - show advanced options > custom index pattern ID : « **trafficrennes-indexpatternid-x9y7z6** », on utilise le même id que celui indiqué dans le fichier de sauvegarde du dashboard (.ndjson)
 - create index pattern

Step 2 of 2: Configure settings

traffic_rennes

Select a primary time field for use with the global time filter.

Time field Refresh

record_timestamp

▼ Hide advanced options

Custom index pattern ID

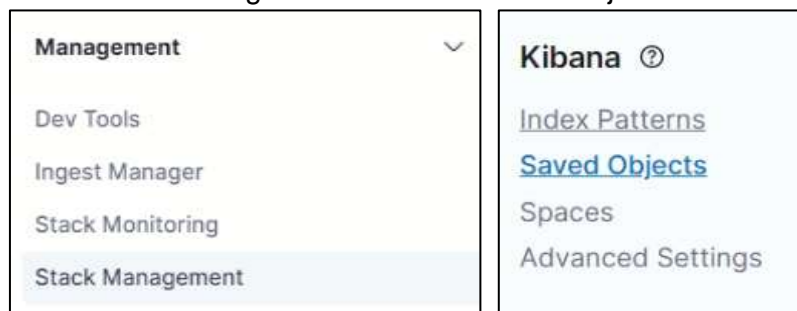
trafficrennes-indexpatternid-x9y7z6

Kibana will provide a unique identifier for each index pattern. If you do not want to use this unique ID, enter a custom one.

Screen 5 – créer un index-pattern (3/3)

Charger le dashboard et ses composantes : on va importer l'objet dashboard dont les informations de configuration stockées dans le fichier de sauvegarde du dashboard. A réaliser aussi depuis kibana :

- Menu management > stack management : kibana > saved objects :



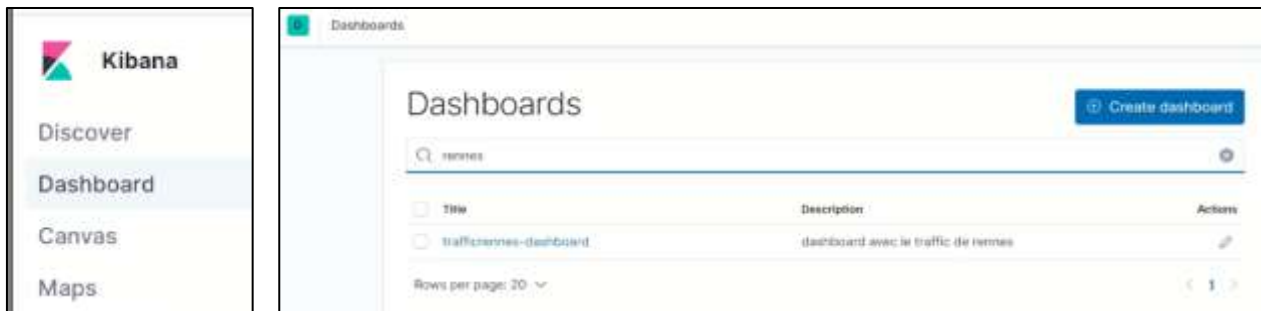
Screen 6 – accéder au menu saved objects

- import : sélectionner le fichier de sauvegarde du dashboard « trafficrennes_dashboard.ndjson »



Screen 7 – importer un objet kibana

- le dashboard est importé et porte le nom « trafficrennes-dashboard »,
 - disponible depuis le Menu kibana > dashboard



Screen 8 – liste des dashboards

5.2 Utilisation

Prérequis : serveur elasticsearch et kibana actifs, avec le dashboard créé/importé.

Programme disponible sur git, et comprend :

- programme batch (.bat) : lance le programme python ci-dessous
- programme python (.py) : réalise le transfert des données
- script python utils (.py) : contient les fonctions créées par les développeurs
- fichier des paramètres (.txt) : pour les paramètres de configuration
- ~~script docker compose (.yaml) : pour la création d'une stack sous docker~~
- ~~fichier de sauvegarde du dashboard (.ndjson) : permet d'importer le dashboard kibana~~

Note : les fichiers ne sont pas utiles pour l'utilisation du programme.

5.2.1 Avec la configuration par défaut

L'utilisation du programme avec la configuration par défaut nécessite que l'ensemble des 4 fichiers se trouvent à la même racine. Ensuite il suffit de lancer le programme via le fichier batch.

Note : le programme nécessite aussi d'avoir créé la variable d'environnement « python » (pointe vers l'exécutable de python). Si ce n'est pas le cas, il suffit de modifier le fichier batch (via un éditeur de texte) et changer la valeur de la variable « python_exe » pour qu'elle pointe vers l'exécutable python.

Par exemple : « python_exe=C:\Users\formation\AppData\Local\Programs\Python\Python39\python.exe ».

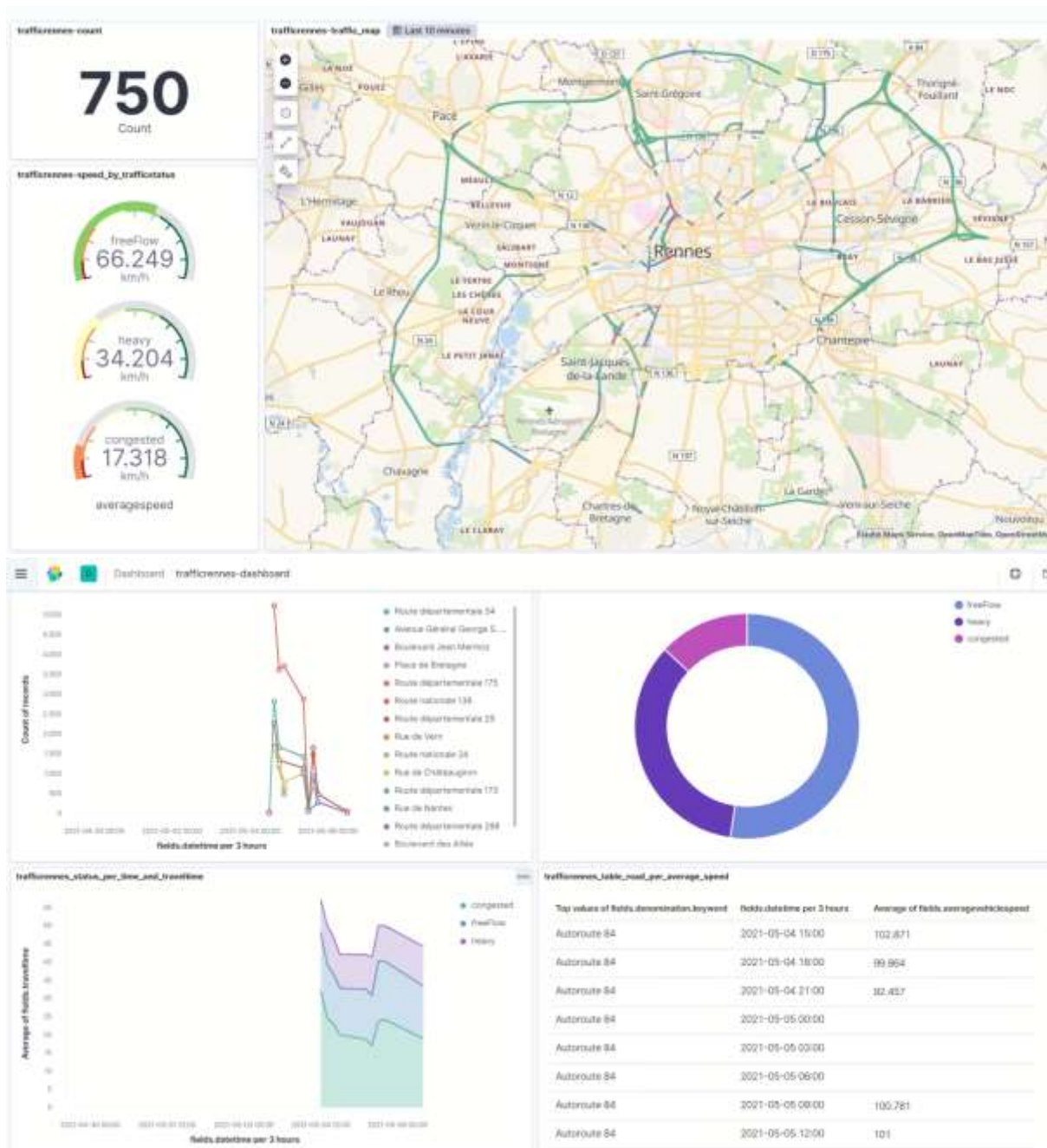
5.2.2 Avec une configuration personnalisée


Depuis le fichier batch :

Depuis le fichier des paramètres :

5.2.3 La log

6 Démo



 Transfert data from traffic api to elasticsearch (python)

```
- Qualité des données
--> reliability...
---> nombre de documents restants: 719/1000 (281 supprimés)

- Export vers elasticsearch
--> export en cours...
---> nombre de documents exportés: 719/719

(--- attente-de-180s-avant-nouvel-appel-api ---)

-- Requête 37/40 ---
- lancement: 2021/05/06 14:23:03

- API traffic
--> requête ok!
--> nombre de documents importés: 1000/1000

- Qualité des données
--> reliability...
---> nombre de documents restants: 731/1000 (269 supprimés)

- Export vers elasticsearch
--> export en cours...
---> nombre de documents exportés: 731/731

(--- attente-de-180s-avant-nouvel-appel-api ---)
```