

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	Dossier d'Architecture Technique.....	3
1.1	Sujet.....	3
1.2	Représentation fonctionnelle	3
1.3	Représentation applicative	3
1.4	Représentation technique / architecture	4
1.4.1	Solution finale : python.....	4
1.4.2	Autre solution : logstash	6
1.4.3	Evolution de la solution : python / flask.....	7
2	Spécification fonctionnelle générale	8
3	Cahier de recette.....	9
4	Manuel	10
4.1	Installation.....	10
4.1.1	Prérequis	10
4.1.2	Serveur elastic	10
4.1.3	Dashboard kibana	10
4.2	Utilisation	12
5	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

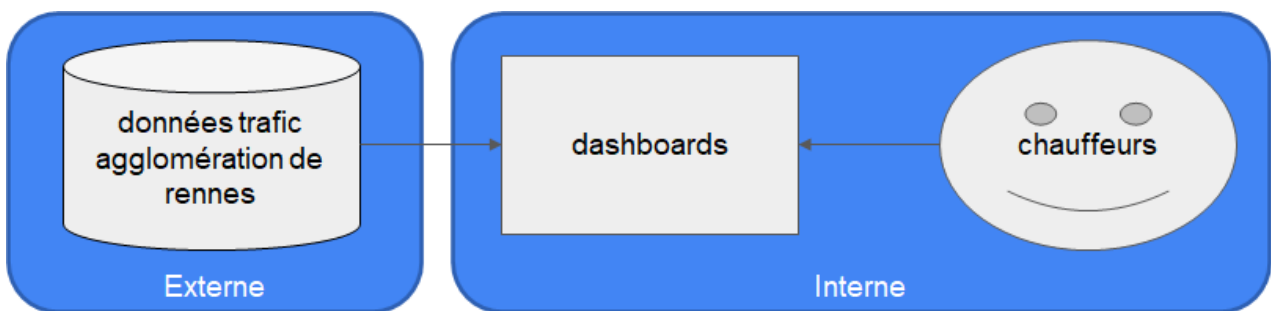


Schéma 1 – représentation fonctionnelle de la demande

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

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

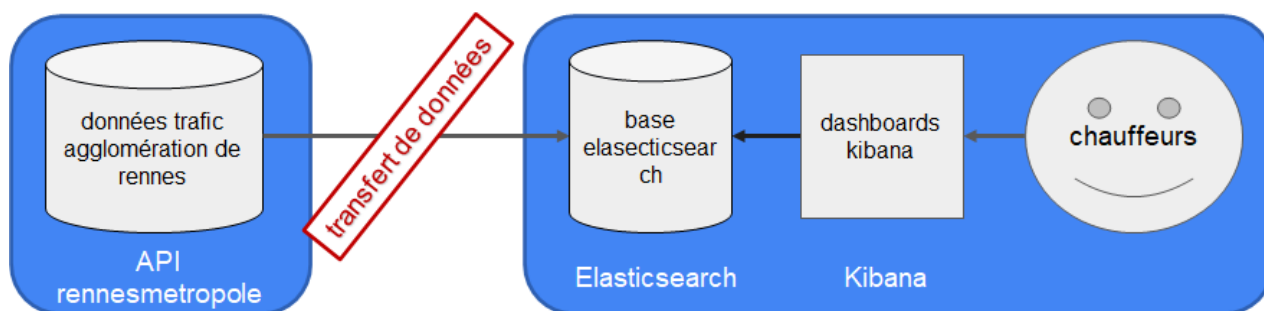


Schéma 2 – représentation applicative de la demande

Ayant plus d'information sur les outils à utiliser, on définit ici la représentation applicative de la demande. On sait sur quelle source de données s'appuyer, et sur quels outils en interne on a à disposition pour le traitement de ces données après leur import.

2.3 Représentation technique / architecture

2.3.1 Solution finale : python

Après la non-validation de la solution logstash (cf. ci-dessous), l'utilisation de python pour fournir un terrain d'entente avec elasticsearch nous paraît un avantage. L'utilisation de python est légère et sa librairie elasticsearch la possibilité de paramétrer la configuration de manière aussi diverse que logstash. En plus des possibilités illimitées de python notamment dans la transformation de données.

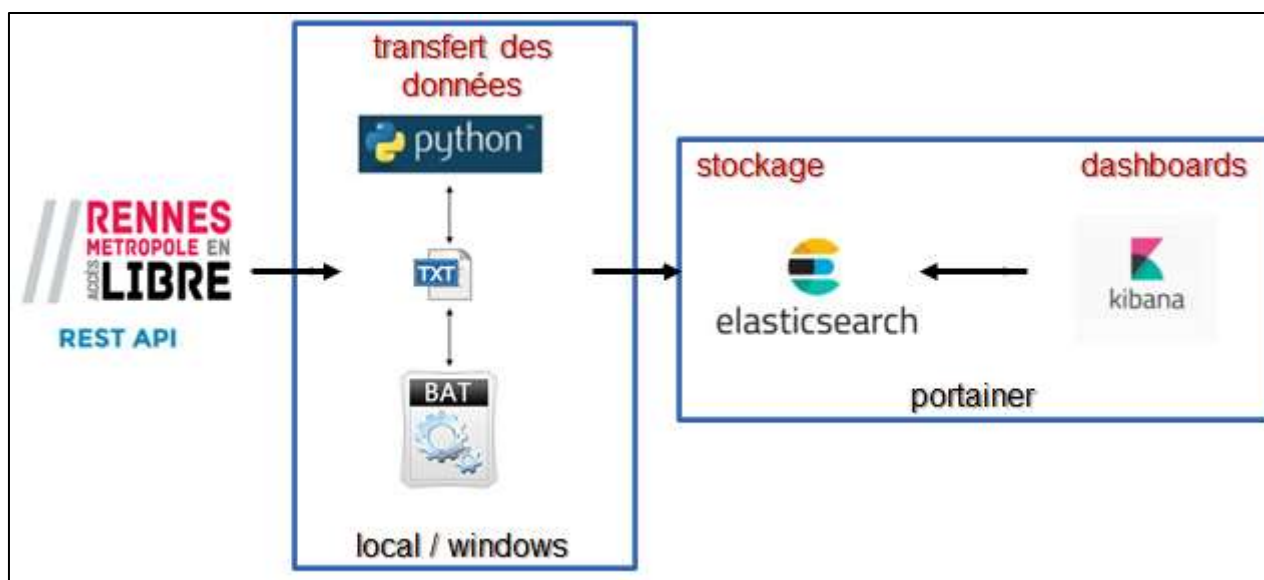


Schéma 3 – représentation de l'architecture technique de la solution

En premier lieu, la création d'un programme python qui va réaliser le transfert des données. Celui-ci comprend :

1. l'appel à l'api et la récupération des données en json
2. le nettoyage et transformations de ces données (détails ci-dessous)
3. l'envoi 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

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).

2.3.2 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.

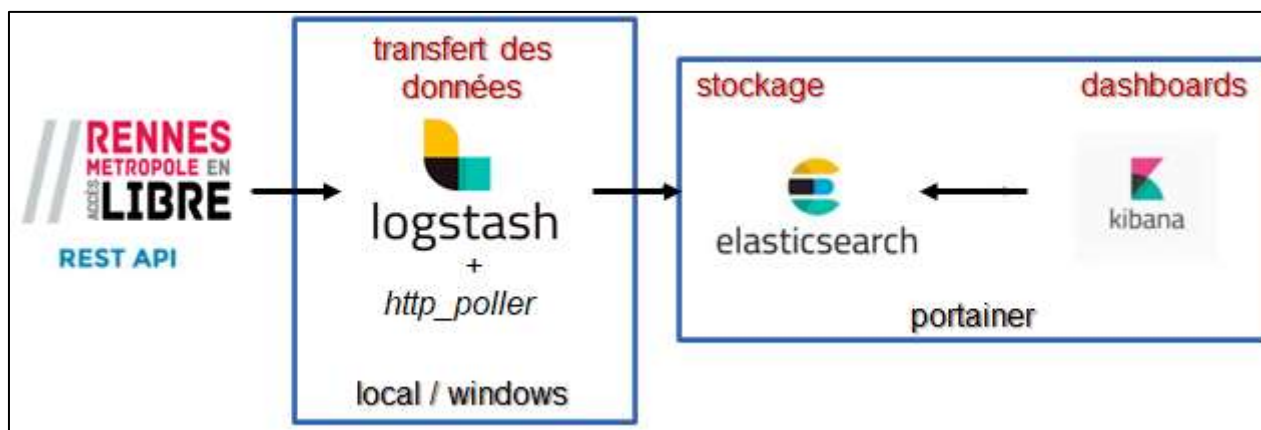


Schéma 4 – représentation de l'architecture logstash

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.

On a décidé d'abandonner cette solution car le mapping automatique des données ne nous a pas donné de résultat utilisable (champs géospatiales non reconnus, cf. Spécification fonctionnelle générale).

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.

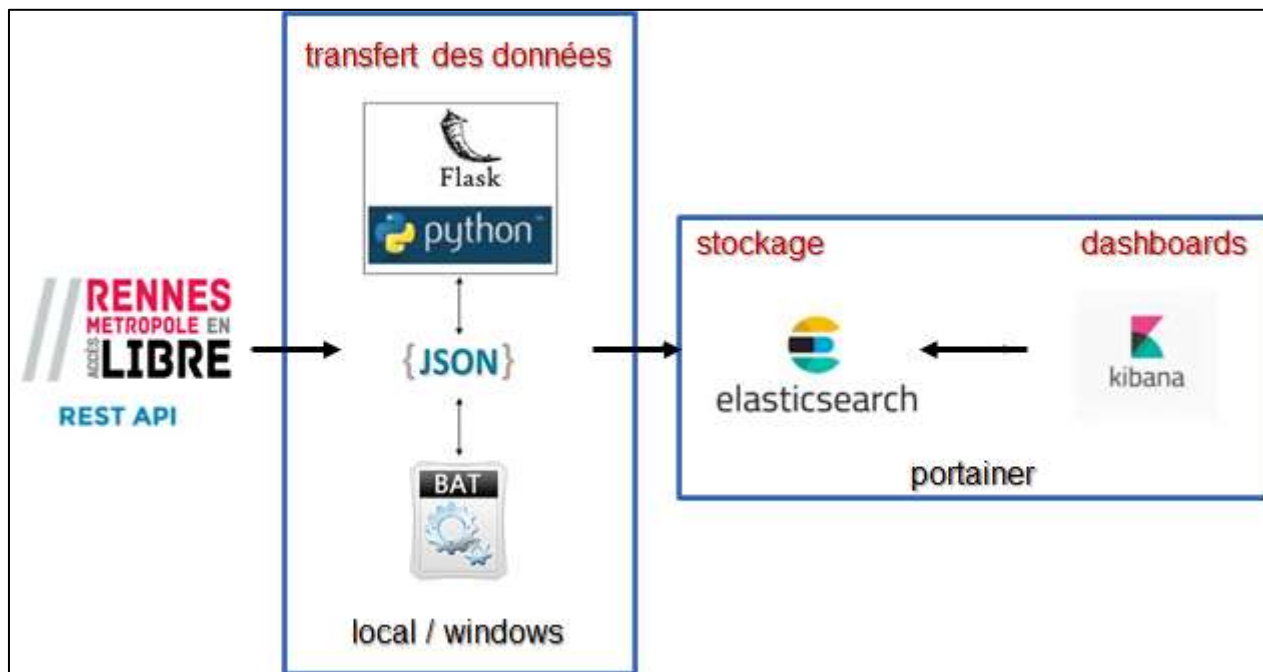


Schéma 5 – architecture de la solution avec python et flask

Il nous permet de construire un formulaire plus intuitif pour sauvegarder les paramètres de configuration (plutôt que d'éditer le fichier texte, cf. Solution finale). 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 :

- 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'envoi 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 :

- 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 où l'on ne garde que les données répondant au critère
 - « *traveltimereliability* >= 50 » : pour la fiabilité des données (cf. 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 d'un sous-champs « *coordinates* » (et non « *type* » et « *coordinates* »), d'où la nécessité déclarer directement ce 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
 - 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 (.yaml)
 - données de création du dashboard (.ndjson)

5.1.2 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).




5.1.3 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 :



The screenshot shows the Kibana Management sidebar on the left with 'Stack Management' selected. On the right, the 'Kibana' menu is open, showing 'Index Patterns' as the first option. Below the menu, the text 'create index pattern' is displayed.

pattern

- create index



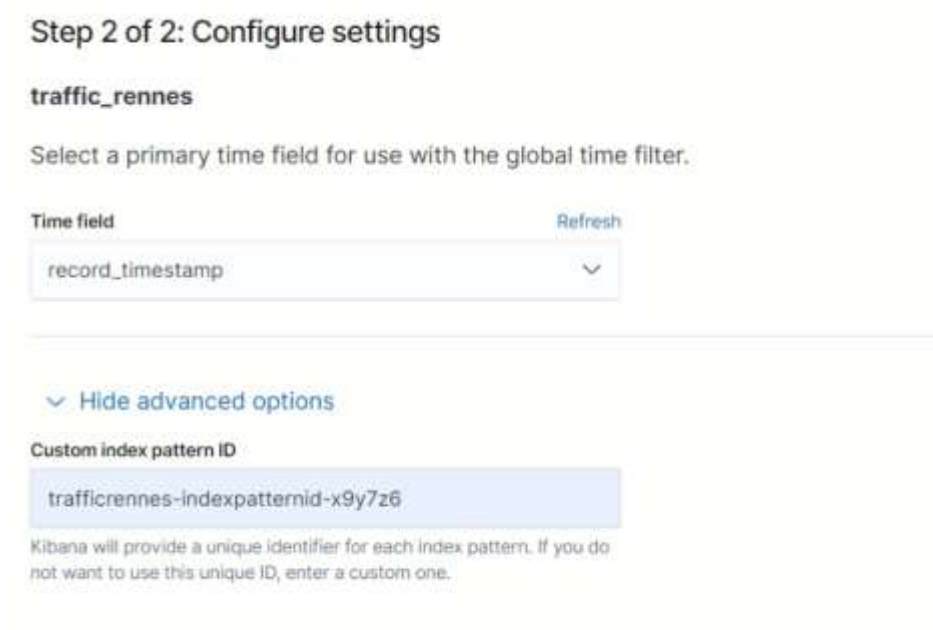
A search bar for 'Index patterns' with a magnifying glass icon and a 'Search...' placeholder. To the right is a blue button labeled 'Create index pattern'.

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



The 'Create index pattern' wizard, Step 1 of 2: Define index pattern. It includes a description of index patterns and a text input field for the 'Index pattern name' with the placeholder 'index-name-*'. A 'Next step >' button is on the right.

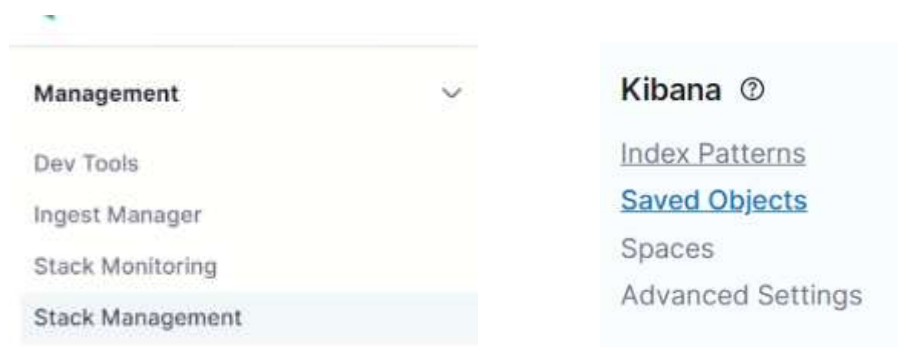
- step 2 of 2 :
 - time field : "fields.datetime", il s'agit ici du champ de filtre temporel
 - show advanced options > custom index pattern ID : "trafficrennes-indexpatternid-x9y7z6", on utilise le même id que celui importé
 - create index pattern



The 'Create index pattern' wizard, Step 2 of 2: Configure settings. It shows the 'traffic_rennes' index pattern. Under 'Time field', 'record_timestamp' is selected. Under 'Custom index pattern ID', 'trafficrennes-indexpatternid-x9y7z6' is entered. A 'Next step >' button is at the bottom right.

Charger le dashboard et ses composantes (kibana):

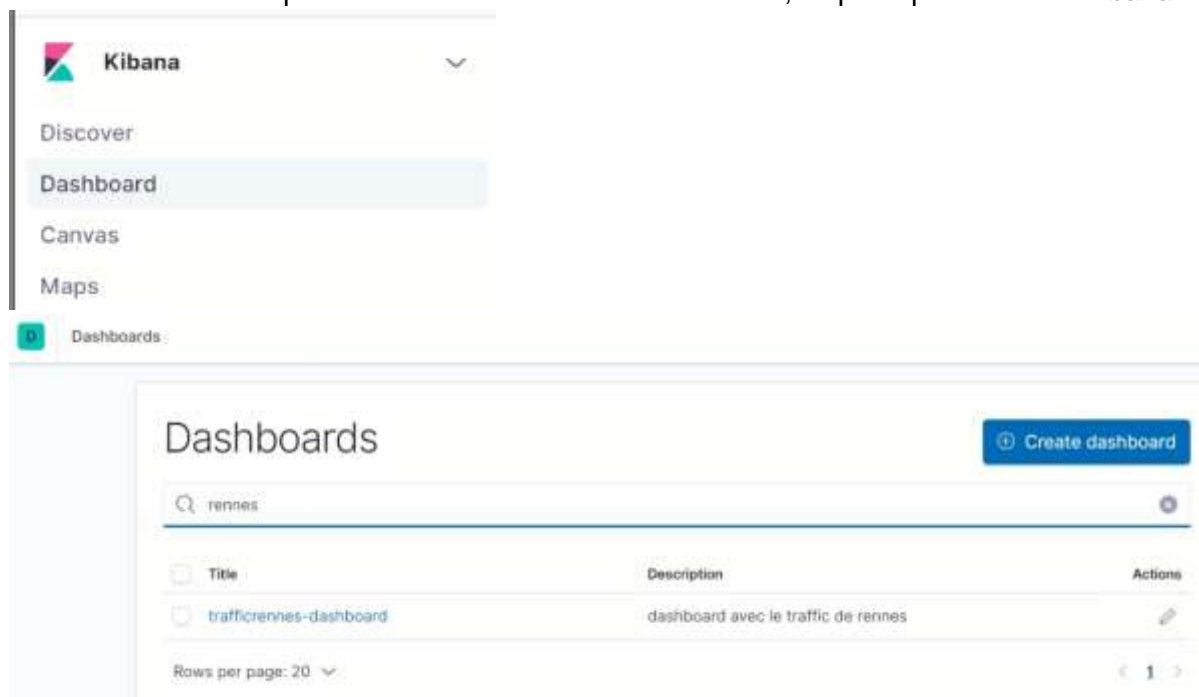
Menu management > stack management : kibana > saved objects :



- import : sélectionner le fichier “trafficrennes_dashboard.ndjson”



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



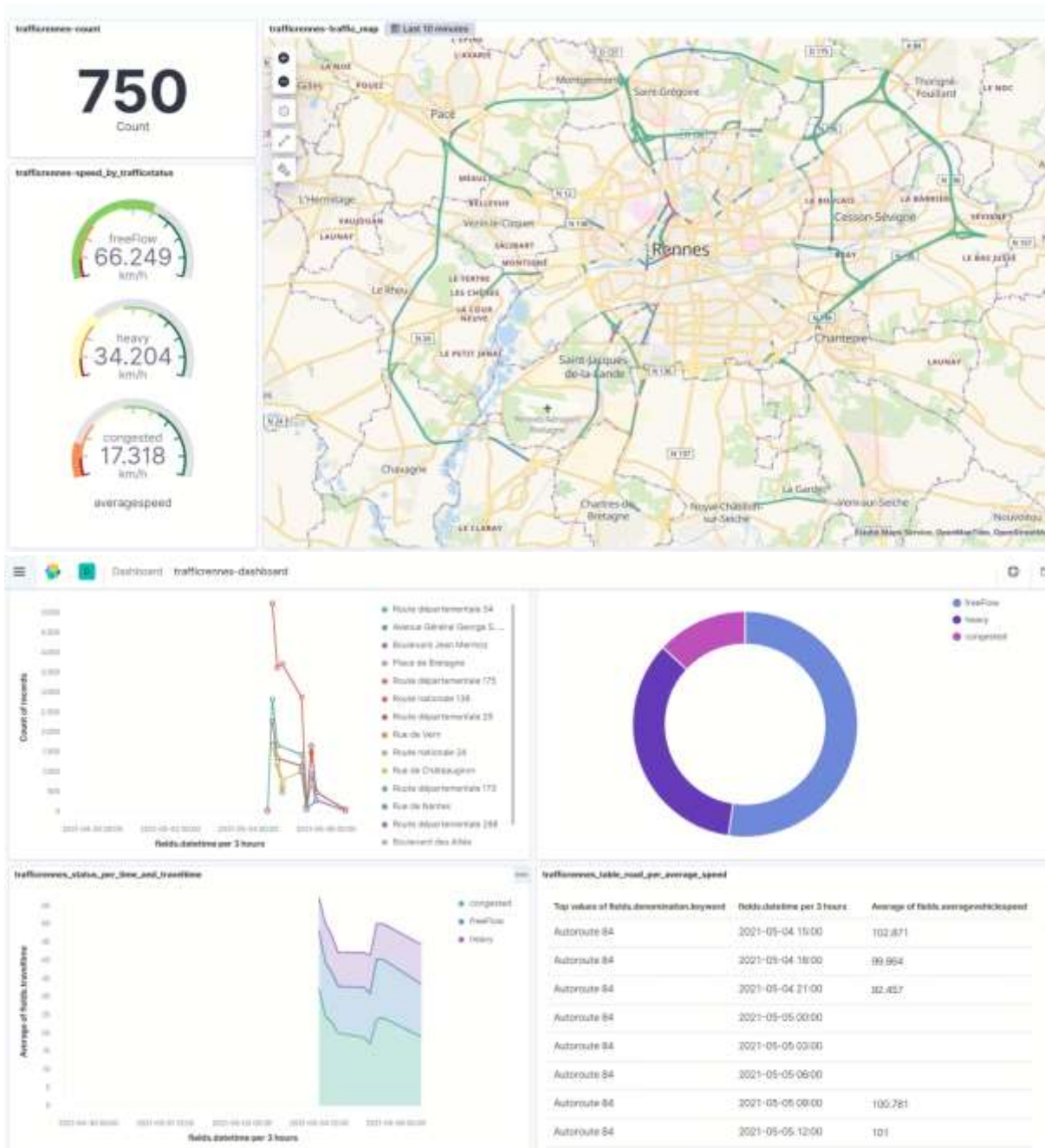
5.2 Utilisation


Prérequis : serveur elasticsearch et kibana actifs.

Par défaut :
Les fichiers

Personnalisé :

6 D  mo



 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 ---)
```