

# DEVELOPPEMENT D'UNE INTERFACE DE GESTION DES LOGS DANS DOWNTIME MANAGER

---

Entreprise SNCF



Rapport de stage

Thylane Rossi

## REMERCIEMENTS

Pour commencer, je tiens d'abord à remercier **Fanny ROSSIGNOL**, qui m'a grandement aidée à trouver ce stage.

Merci à **Bertrand DUFOUR**, mon tuteur de stage et responsable de domaine (RDA), qui a accepté de me prendre en stage et de m'avoir permise de travailler sur un sujet qui correspond parfaitement aux compétences que j'ai développées au sein de mes trois années de licence.

Merci à **Abderrahmane TADJ** et **Thomas THEROUANNE**, mes tuteurs techniques et OPS au sein de l'équipe Géoservices, qui sont à l'origine de ce projet et qui répondent à mes questions, prennent le temps de m'expliquer et me donnent des conseils pour m'améliorer.

Merci à tous les membres de l'équipe GEOSERVICES avec qui j'ai pu échanger, apprendre et progresser grâce à leurs conseils, leurs remarques et leur bienveillance.

Merci également à mes proches pour la relecture de ce rapport.

Enfin, merci à l'Université Lyon 1 d'intégrer dans son cursus l'opportunité de stage qui permet de mettre en pratique mes compétences techniques au sein d'une entreprise.

## TABLE DES MATIERES

Remerciements .....	1
Table des figures .....	3
Glossaire .....	4
Introduction .....	5
Description de l'entreprise .....	6
La SNCF et mon service d'affectation.....	6
Les différentes applications GEOSERVICES .....	7
GEOMOBILES.....	7
GEOMAPS.....	7
GEOPULSE.....	8
GEO2RAIL .....	8
GEONOTIF .....	9
GPSTRAINS .....	9
Organisation du travail et outils utilisés .....	10
Organisation du travail et de l'équipe .....	10
Outils de développement .....	10
L'application et le travail réalisé .....	10
Mise en contexte - Présentation de Downtime Manager .....	10
L'architecture du projet.....	14
L'origine des logs.....	15
Emission des positions des trains.....	15
Coût des logs .....	16
Le travail réalisé .....	18
L'affichage général des environnements .....	18
L'affichage des détails de chaque ressource .....	19
Optimisation du temps de chargement des données .....	20
Modification de la rétention.....	21
La suite du stage.....	23
Modification des logs envoyés à Datadog.....	23
Restructuration du code avec une classe.....	23
Tests unitaires pour un déploiement de la nouvelle version .....	23

Présentation de l'application aux autres services SNCF.....	24
Conclusion.....	24
Annexe.....	25

## TABLE DES FIGURES

FIGURE 1 - ORGANIGRAMME SNCF .....	6
FIGURE 2 – INTERFACE DE L'APPLICATION GEOMOBILES .....	7
FIGURE 3 - INTERFACE DE L'APPLICATION GEOMAPS .....	7
FIGURE 4 - INTERFACE DE L'APPLICATION GEOPULSE .....	8
FIGURE 5 - INTERFACE DE L'APPLICATION GEO2RAIL.....	8
FIGURE 6 - INTERFACE DE L'APPLICATION GEONOTIF .....	9
FIGURE 7 - BALISE GPSTRAINS .....	9
FIGURE 8 - INTERFACE DE LA MODIFICATION DES ANNOTATIONS BASES DE DONNEES RDS .....	11
FIGURE 9 - INTERFACE DE LA MODIFICATION DES ANNOTATIONS KUBERNETES .....	12
FIGURE 10 - INTERFACE POUR VOIR L'ETAT DES ENVIRONNEMENTS .....	12
FIGURE 11 - INTERFACE POUR GERER L'ETAT DES ENVIRONNEMENTS .....	13
FIGURE 12 - SCHEMA SIMPLIFIE DE L'ARCHITECTURE DU PROJET .....	14
FIGURE 13 - SCHEMA D'EMISSION DES POSITIONS .....	15
FIGURE 14 - EXTRAIT DES LOGS DU DEPLOY GOM-SERVICE-ROUTER SUR 24H .....	16
FIGURE 15 - TABLEAU DE SUIVI DE LA CONSOMMATION MENSUELLE EN FONCTION DE LA RETENTION	17
FIGURE 16 - PRIX DE VENTE DES UO DATADOG 2025 .....	17
FIGURE 17 - INTERFACE D'AFFICHAGE DES ENVIRONNEMENTS .....	18
FIGURE 18 - INTERFACE DU DETAIL DE CHAQUE RESSOURCE D'UN ENVIRONNEMENT (ICI GEOMOBILES – INTEGRATION).....	19
FIGURE 19 - FONCTION PRELOAD_ALL_DATA() DU FICHIER APP.PY QUI LANCE LE CHARGEMENT DES DONNEES EN PARALLELE .....	20
FIGURE 20 - EXTRAIT DU TERMINAL LORS DU CHARGEMENT DES LOGS.....	21
FIGURE 21 - RESUME DES MODIFICATIONS DES RETENTIONS DES DEPLOIEMENTS .....	22
FIGURE 22 - BOUT DE CODE DE LA SOCKET POUR LA MODIFICATION DE LA RETENTION .....	22

## GLOSSAIRE

**DevOps** : Démarche visant, par la collaboration entre toutes les parties prenantes d'un projet informatique, depuis le développement jusqu'aux opérations, à rendre l'entreprise dans son ensemble plus flexible et communicante.

**OPS** : Personnes responsables de la gestion de l'infrastructure, des réseaux et des systèmes qui soutiennent le processus de développement de logiciels

**GNSS (Global Navigation Satellite Systems)** : Système de positionnement par satellites.

**Tablette SIRIUS** : Tablette à disposition des conducteurs qui fournit sa position et une aide à la conduite.

**API (interface de programmation d'application)** : Interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

**Numéro d'engin** : Numéro associé à une locomotive.

**Course** : Trajet commercial effectué par un train.

**JSON** : (JavaScript Object Notation) Format d'échange de données en texte lisible.

**Environnement** : Contexte d'exécution d'une application (assemblage, intégration, recette, préproduction, production). Chaque environnement a ses propres réglages et ressources.

**Ressource** : Objet Kubernetes, comme un Deployment ou un StatefulSet, déployé dans un environnement. C'est une brique technique qui fait tourner l'application.

**Annotation Kubernetes** : Clé/valeur ajoutée à un objet Kubernetes pour fournir des infos utiles à d'autres outils (ex : durée de rétention des logs).

**Base de données RDS** : Base de données gérée par AWS.

**Datadog** : Plateforme de supervision et de monitoring des systèmes d'information. Elle permet de collecter, visualiser et analyser en temps réel les métriques, logs et traces des applications. À la SNCF, Datadog est utilisé pour surveiller l'état de santé des applications, détecter les anomalies et faciliter l'analyse des incidents.

## INTRODUCTION

Dans le cadre de ma troisième année de licence Informatique à l'Université Claude Bernard Lyon 1, j'ai effectué mon stage au sein de la SNCF. Ce stage s'est inscrit dans une démarche d'optimisation des ressources informatiques, avec un focus particulier sur la maîtrise des coûts liés à la gestion des logs.

Mon travail s'intègre dans une application existante appelée Downtime Manager, un outil interne utilisé pour la gestion des environnements Kubernetes. Cette application permet aux équipes de planifier automatiquement l'arrêt ou le démarrage des environnements en fonction des plages horaires souhaitées, en s'appuyant sur des annotations Kubernetes.

Ma mission principale consiste à enrichir l'application Downtime Manager en développant une interface permettant de visualiser, modifier et contrôler les paramètres des logs envoyés vers Datadog. Cette interface vise à rendre plus accessible l'ajustement des règles d'ingestion et d'indexation, des opérations qui sont aujourd'hui techniques, peu ergonomiques, et par conséquent, rarement réalisées par les équipes.

L'envoi et la conservation de logs dans Datadog représentent un coût financier et environnemental significatif, qui peut rapidement s'accroître lorsque des logs peu utiles sont collectés ou conservés trop longtemps. De plus, cette surconsommation de ressources numériques contribue directement à une forte empreinte environnementale.

Dans ce rapport, je présenterai dans un premier temps l'environnement dans lequel j'ai évolué, l'entreprise SNCF ainsi que l'équipe avec laquelle je travaille. J'aborderai ensuite le fonctionnement de l'équipe et les outils utilisés au quotidien. Puis, je présenterai en détail le projet Downtime Manager, son architecture et son rôle dans la gestion des environnements. La dernière partie sera consacrée à l'explication technique du travail que j'ai réalisé jusqu'à présent, ainsi qu'aux évolutions prévues jusqu'à la fin du stage, fixée au 8 août.

## DESCRIPTION DE L'ENTREPRISE

### LA SNCF ET MON SERVICE D'AFFECTATION

Depuis 2020, SNCF SA est la maison-mère du Groupe SNCF et pilote ses orientations stratégiques, budgétaires et industrielles. J'ai effectué mon stage au sein de la Direction de l'Exploitation Applicative (DEA), dirigée par Bertrand GOUVERNEUR.

Plus précisément, j'ai travaillé dans la Direction des Services Métiers – Production Ferroviaire / Personnel Roulant (DSM PF / PR), encadrée par Laurent COULET. Mon équipe fait partie de e.SNCF, rattachée à la branche STIV (Supervision Transporteurs et Informations Voyageurs), dirigée par Bertrand DUFOUR.

e.SNCF Solutions propose un ensemble de services pour construire, intégrer et exploiter les applications du SI SNCF. Ces services couvrent l'hébergement, l'infrastructure, les outils techniques, ainsi que l'exploitation applicative.

e.SNCF Solutions propose une offre complète de services pour accompagner les équipes SNCF dans la construction, l'intégration et l'exploitation de leurs applications. Cette offre couvre plusieurs domaines clés : l'hébergement (OnPremise ou cloud), l'infrastructure (stockage, sauvegarde, serveurs), l'outillage technique (ordonnancement, supervision, conteneurisation), ainsi que l'exploitation et la maintenance applicative.

Parmi les bénéficiaires de ces services figure **Géoservices**, un client interne qui s'appuie sur cette infrastructure pour faire fonctionner ses applications métier.





## LES DIFFERENTES APPLICATIONS GEOSERVICES

L'application sur laquelle j'ai travaillé s'inscrit dans une démarche d'amélioration de la gestion des environnements et de la supervision des logs pour les différentes applications de Géoservices.

### GEOMOBILES

**GEOMOBILES** récupère en temps réel la position des trains à partir de plusieurs sources comme les balises GNSS, les tablettes SIRIUS ou les données réseau. Ces positions sont enrichies avec un contexte géographique et horaire, puis redistribuées à différents services internes de SNCF Voyageurs.

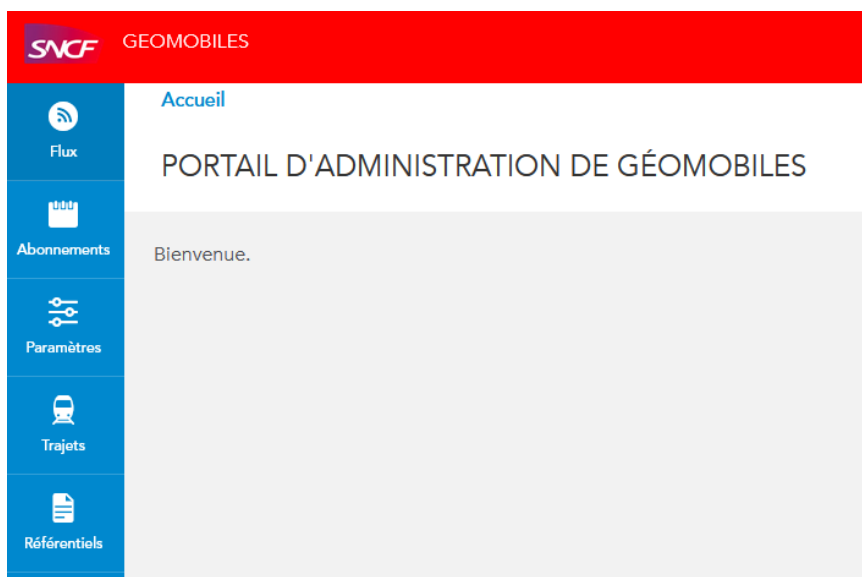


Figure 2 – Interface de l'application Geomobiles

### GEOMAPS

**GEOMAPS** propose des fonds de carte OpenStreetMap personnalisés, optimisés pour mettre en valeur les infrastructures ferroviaires. L'application fournit également des objets vectoriels issus du référentiel RGI de SNCF Réseau.

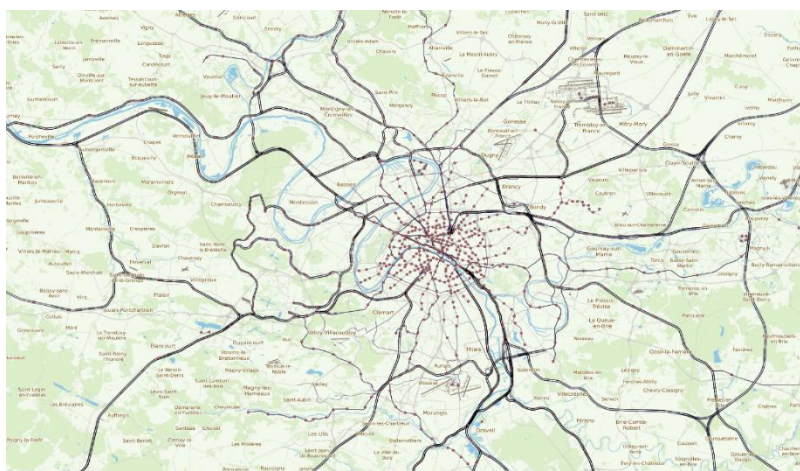


Figure 3 - Interface de l'application



## GEOPULSE

**GEOPULSE** permet de visualiser en temps réel les circulations ferroviaires sur une carte. Elle est largement utilisée dans les centres opérationnels pour suivre les trains, en combinant les données de GEOMOBILES avec les fonds cartographiques de GEOMAPS.

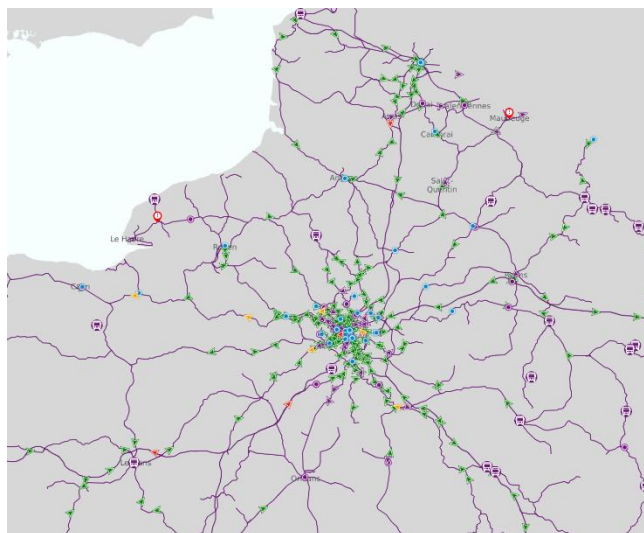


Figure 4 - Interface de l'application Geopulse

## GEO2RAIL

**GEO2RAIL** effectue une cartospondance, c'est-à-dire qu'il génère le tracé précis d'un trajet sur les voies ferrées à partir d'une liste de points. Il s'appuie sur GraphHopper et peut utiliser OpenStreetMap ou le référentiel GAIA selon les besoins.



Figure 5 - Interface de l'application Geo2rail

## GEONOTIF

**GEONOTIF** transforme les données GPS des trains en événements concrets, comme des alertes d'entrée ou de sortie de zone. Cela permet une meilleure automatisation du suivi des circulations et des prises de décision.

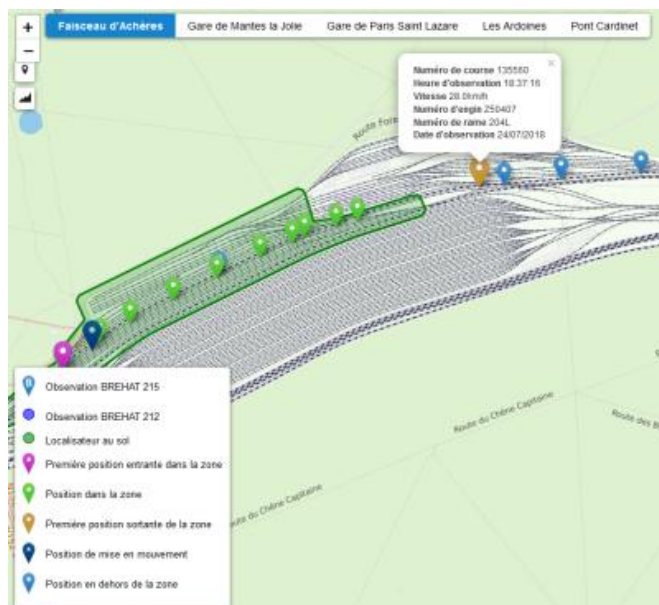


Figure 6 - Interface de l'application Geonotif

## GPSTRAINS

**GPSTRAINS** sert à géolocaliser les locomotives équipées de balises GPS autonomes. Il permet notamment de suivre les engins et de calculer les kilomètres parcourus afin de planifier leur maintenance.



Figure 7 - Balise GPSTRAINS

## ORGANISATION DU TRAVAIL ET OUTILS UTILISES

### ORGANISATION DU TRAVAIL ET DE L'EQUIPE

L'équipe suit un rythme de réunions quotidiennes à 9h30, du lundi au jeudi. Chaque membre y présente brièvement ses tâches prévues pour la journée. Le vendredi, une réunion plus complète est tenue à 9h, réunissant les OPS et les chefs de projet pour faire un point d'avancement sur l'ensemble des applications de Géoservices, lever les éventuels blocages et assurer une coordination globale.

L'équipe applique également la méthodologie Agile, avec des cycles de travail de quatre semaines. À la fin de chaque itération, un point collectif est organisé pour faire le bilan, ajuster les priorités et planifier les actions à venir..

En dehors des réunions, les échanges se font principalement via Microsoft Teams, utilisé pour la communication informelle, le partage de ressources, la coordination des tâches et les appels à distance.

Le stage se déroule au 14e étage de la Tour Oxygène à Lyon, dans un open space partagé avec d'autres équipes SNCF.

### OUTILS DE DEVELOPPEMENT

Le développement de l'application est réalisé principalement dans Visual Studio Code, en local sur un environnement Python exécuté sous WSL dans une machine virtuelle. Le backend repose sur Flask, avec Socket.IO pour les communications en temps réel et Jinja2 pour le rendu côté serveur. L'interface est construite en JavaScript et HTML.

Une partie des données utilisées provient d'un inventaire externe structuré en JSON, qui sert à organiser les environnements, les applications et leurs configurations. Le projet s'appuie sur des technologies comme Kubernetes et AWS pour la gestion des environnements. L'ensemble du travail est versionné sur Git, une branche est dédiée pour le développement de ma fonctionnalité poc\_datadog. Les OPS supervisent ensuite la revue de code, valident les modifications, puis fusionnent la branche après test.

## L'APPLICATION ET LE TRAVAIL REALISE

### MISE EN CONTEXTE - PRESENTATION DE DOWNTIME MANAGER

Mon travail s'inscrit dans une application déjà existante appelée Downtime Manager, développée par les deux OPS avec qui je travaille, Abderrahmane TADJ et Thomas THEROUANNE. Cette application interne a été conçue pour répondre à deux enjeux principaux : d'une part, mieux maîtriser les coûts liés aux ressources cloud, et d'autre part, offrir plus d'autonomie aux développeurs dans la gestion de leurs environnements.

À l'origine, seuls les OPS pouvaient allumer ou éteindre les environnements, ce qui pouvait entraîner des blocages si un développeur avait besoin d'un environnement alors que les OPS n'étaient pas disponibles. Downtime Manager vise donc à rendre cette gestion plus accessible.

Plusieurs interfaces ont été créées dans ce but :

- La première permet de programmer l'arrêt ou le démarrage automatique des bases de données RDS, en définissant une plage horaire pour chaque jour (voir Figure 8).

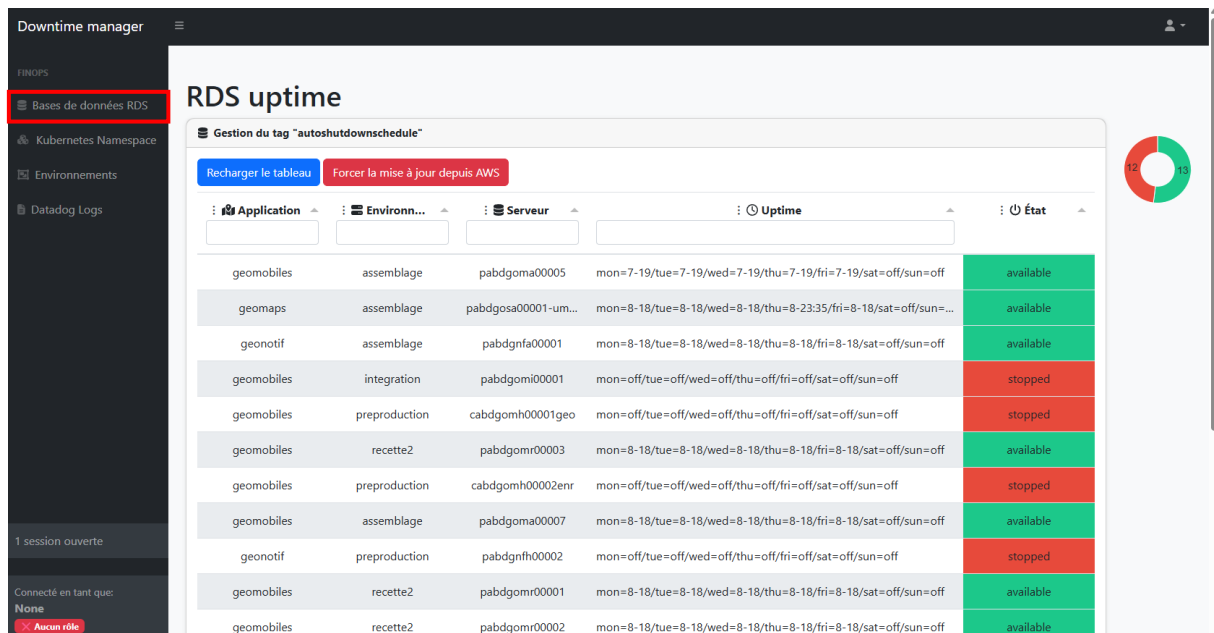
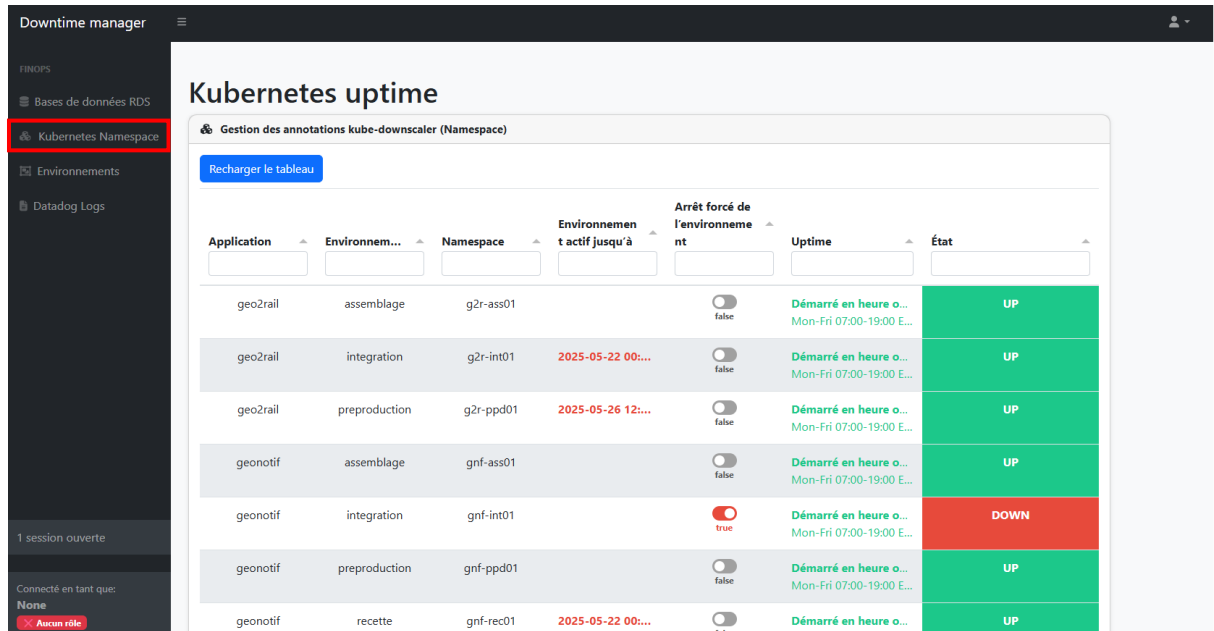


Figure 8 - Interface de la modification des annotations bases de données RDS

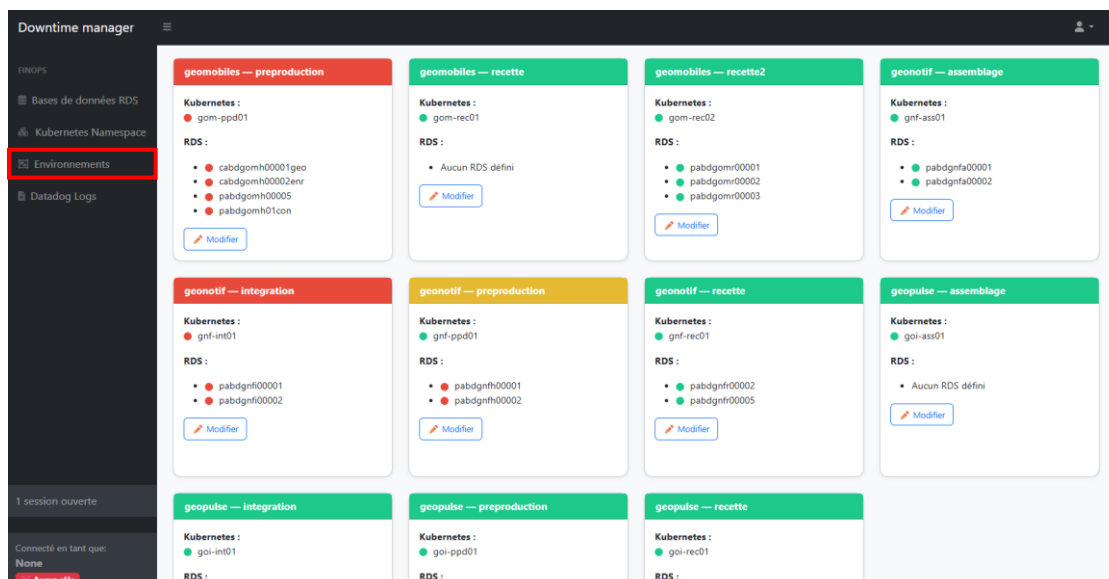
- La deuxième sert à gérer l'état des namespaces Kubernetes, notamment pour planifier leur extinction ou leur démarrage (voir Figure 9).



Application	Environnement	Namespace	Environnement actif jusqu'à	Arrêt forcé de l'environnement	Uptime	État
geo2rail	assemblage	g2r-ass01		false	Démarré en heure o... Mon-Fri 07:00-19:00 E...	UP
geo2rail	integration	g2r-int01	2025-05-22 00:...	false	Démarré en heure o... Mon-Fri 07:00-19:00 E...	UP
geo2rail	preproduction	g2r-ppd01	2025-05-26 12:...	false	Démarré en heure o... Mon-Fri 07:00-19:00 E...	UP
geonotif	assemblage	gnf-ass01		false	Démarré en heure o... Mon-Fri 07:00-19:00 E...	UP
geonotif	integration	gnf-int01		true	Démarré en heure o... Mon-Fri 07:00-19:00 E...	DOWN
geonotif	preproduction	gnf-ppd01		false	Démarré en heure o... Mon-Fri 07:00-19:00 E...	UP
geonotif	recette	gnf-rec01	2025-05-22 00:...	false	Démarré en heure o...	UP

Figure 9 - Interface de la modification des annotations Kubernetes

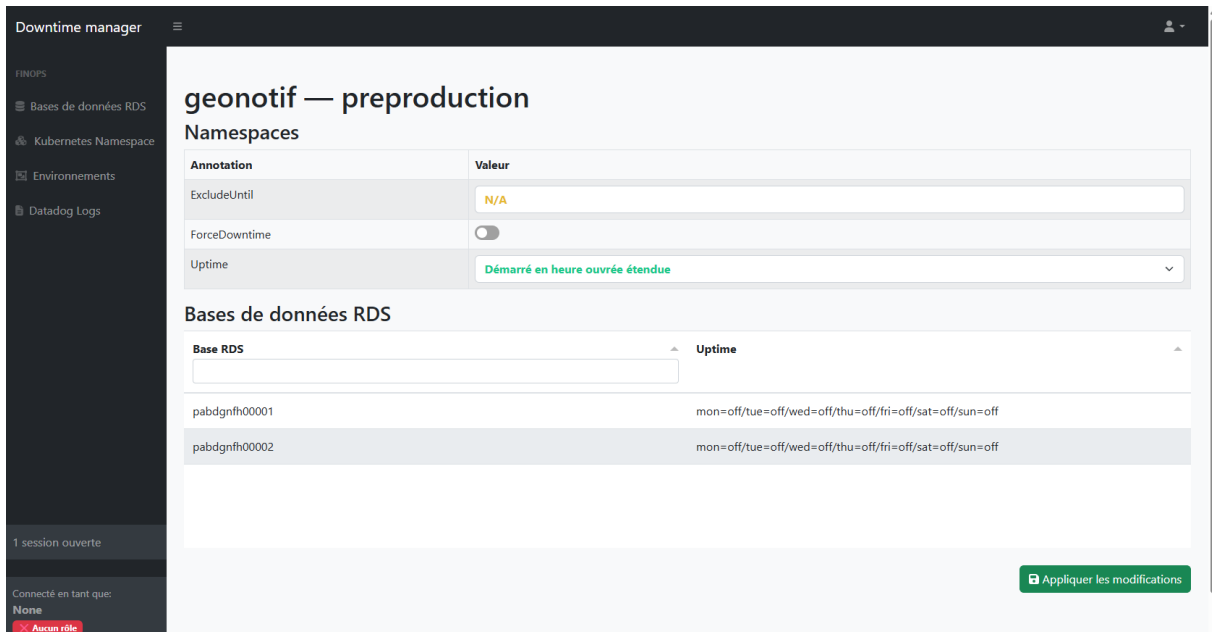
- Une troisième interface regroupe ces informations et affiche, pour chaque environnement, un état global : vert si tout est allumé, rouge si tout est éteint, jaune s'il y a une incohérence entre les deux (voir Figure 10).



Environnement	Kubernetes	RDS
geomobiles — preproduction	gom-ppd01 (red dot)	Aucun RDS défini
geomobiles — recette	gom-rec01 (green dot)	Aucun RDS défini
geomobiles — recette2	gom-rec02 (green dot)	pabdgm00001 (green dot), pabdgm00002 (green dot), pabdgm00003 (green dot)
geonotif — assemblage	gnf-ass01 (green dot)	pabdgnta00001 (green dot), pabdgnta00002 (green dot)
geonotif — integration	gnf-int01 (red dot)	pabdgntf00001 (red dot), pabdgntf00002 (red dot)
geonotif — preproduction	gnf-ppd01 (green dot)	pabdgntf00001 (red dot), pabdgntf00002 (red dot)
geonotif — recette	gnf-rec01 (green dot)	pabdgntf00002 (green dot), pabdgntf00005 (green dot)
geopulse — assemblage	goi-ass01 (green dot)	Aucun RDS défini
geopulse — integration	goi-int01 (green dot)	
geopulse — preproduction	goi-ppd01 (green dot)	
geopulse — recette	goi-rec01 (green dot)	

Figure 10 - Interface pour voir l'état des environnements

- Enfin, une dernière interface accessible en appuyant sur le bouton « modifier » de la troisième interface, permet d'accéder à toutes les options de gestion sur une seule page, pour modifier à la fois les bases de données et les namespaces (voir Figure 11).



The screenshot shows the 'Downtime manager' interface for the 'geonotif — preproduction' environment. The interface is divided into two main sections: 'Namespaces' and 'Bases de données RDS'.

**Namespaces Section:**

Annotation	Valeur
ExcludeUntil	N/A
ForceDowntime	<input type="checkbox"/>
Uptime	Démarré en heure ouvrée étendue

**Bases de données RDS Section:**

Base RDS	Uptime
pabdgafh00001	mon=off/tue=off/wed=off/thu=off/fri=off/sat=off/sun=off
pabdgafh00002	mon=off/tue=off/wed=off/thu=off/fri=off/sat=off/sun=off

At the bottom right, there is a green button labeled 'Appliquer les modifications'.

Figure 11 - Interface pour gérer l'état des environnements

## L'ARCHITECTURE DU PROJET

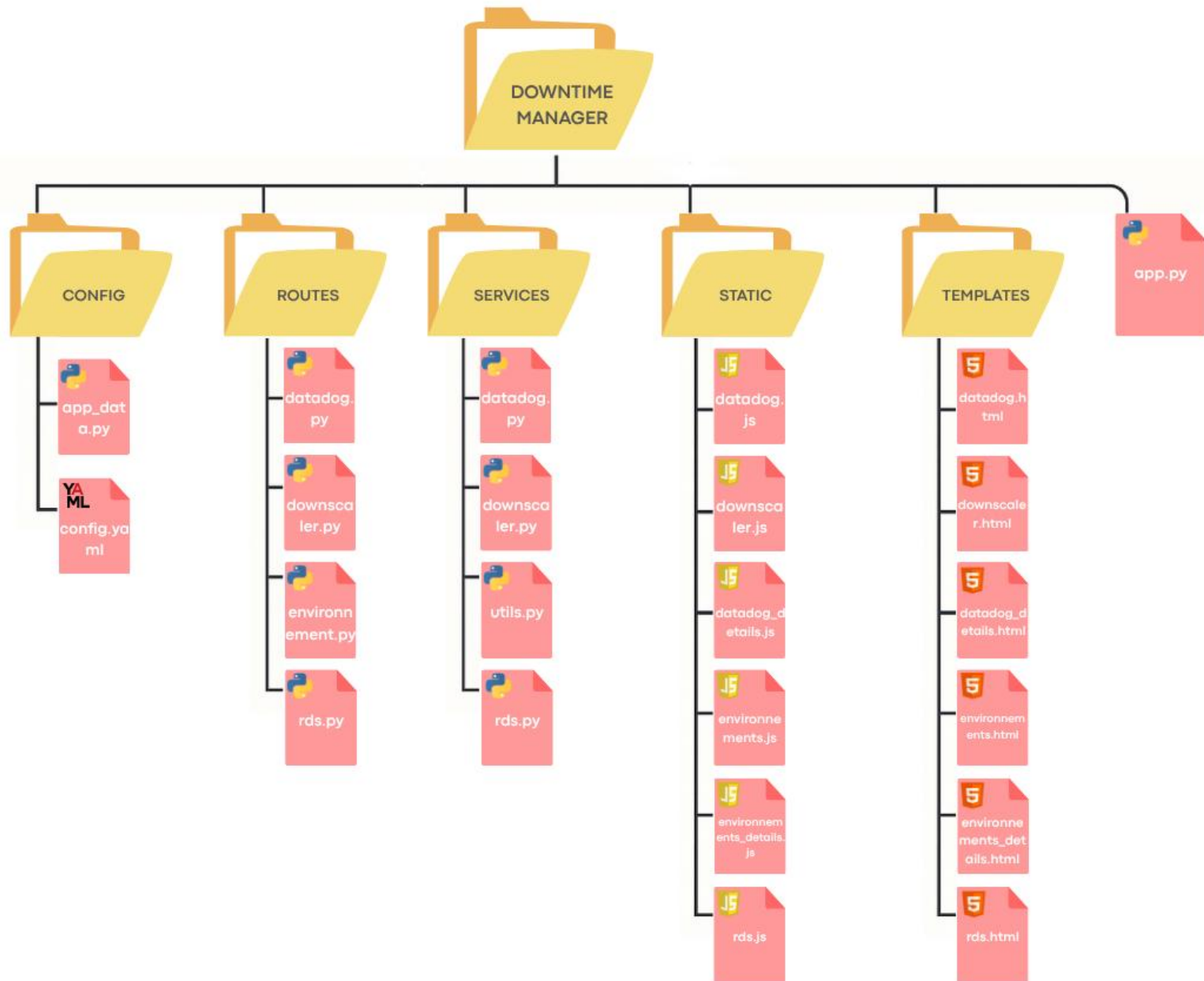


Figure 12 - Schéma simplifié de l'architecture du projet



Voici une explication simplifiée de l'architecture de l'application Downtime Manager. Elle est composée de six éléments principaux : cinq dossiers (config, routes, services, static, templates) et un fichier à la racine app.py.

- app.py est le point d'entrée de l'application. Il initialise le serveur Flask, configure l'authentification, lance les WebSockets et orchestre le chargement des données via des threads parallèles.
- config/ contient les fichiers de configuration. Le fichier app\_data.py de ce dossier centralise les données partagées dans l'application, tandis que config.yaml définit la structure des applications, environnements et namespaces.
- routes/ regroupe les routes Flask de l'application.
- services/ contient la logique métier associée aux différentes fonctionnalités, comme les fonctions pour enrichir les données ou les fonctions de mise à jour.
- static/ contient les fichiers JavaScript frontend qui rendent les interfaces interactives.
- templates/ regroupe les fichiers HTML rendus côté serveur.

## L'ORIGINE DES LOGS

Pour bien comprendre la raison de mon stage, il faut se rendre compte du nombre de logs reçus chaque seconde. La principale source de logs est l'émission des positions des trains.

### EMISSION DES POSITIONS DES TRAINS

Il y a 3 grandes sources d'émissions des données de géolocalisation.

Pour commencer, nous avons la **balise GPS**, elle est positionnée sur le toit des trains elle a des avantages et des limites. Tout d'abord cette balise émet de manière régulière avec une très bonne précision, de plus elle est liée à un train, on peut savoir exactement où est quel train. Cependant, elle possède quelques limites, tout d'abord sa fréquence d'émission peut parfois être très longue, elle émet en moyenne toutes les 45 minutes, elle ne peut pas émettre quand le train est dans un tunnel et pour finir, ces balises ont un coût assez élevé.

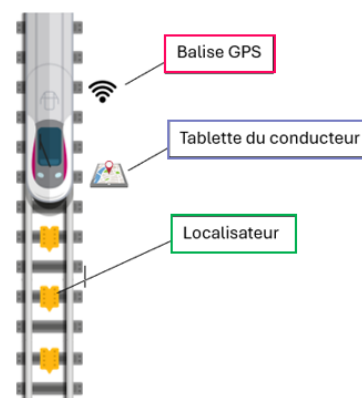


Figure 13 - Schéma d'émission des positions

La deuxième source d'émission de données de géolocalisation est la **tablette du conducteur**. C'est un outil d'aide à la conduite que les conducteurs choisissent d'utiliser. Comme les balises, ces tablettes possèdent des avantages comme une meilleure fréquence d'émission et une bonne

précision. Mais, ces tablettes, n'émettent également pas dans les tunnels, elles ne sont pas liées à un engin et il se peut qu'il y ait des erreurs d'association avec la course car les numéros de course sont rentrés manuellement par le conducteur. De plus, comme dit plus haut, les conducteurs ont le choix d'utiliser ces tablettes, environ 15% ne les utilisent pas ce qui limite une partie des émissions GPS.

La troisième et dernière source d'émission est constituée des **localisateurs**, installés directement sur les rails. Lorsqu'un train passe dessus, ces dispositifs détectent physiquement son passage. Bien qu'ils ne puissent pas identifier formellement le train, ils peuvent en déduire, avec un certain degré de probabilité, le numéro de l'engin et la course qu'il effectue. Cette incertitude constitue leur principale limite. En revanche, les localisateurs présentent l'avantage de couvrir de larges zones et d'offrir une fréquence d'émission élevée.

## COUT DES LOGS

Les envois de ces positions provoquent des logs, on reçoit **environ 200 positions par seconde**, il est donc essentiel de pouvoir les filtrer pour limiter les coûts de stockage. En effet, on peut voir sur la figure 14 que les logs reçus, seulement par le deploy gom-service-router sur 24h, sont de 24,4 millions.

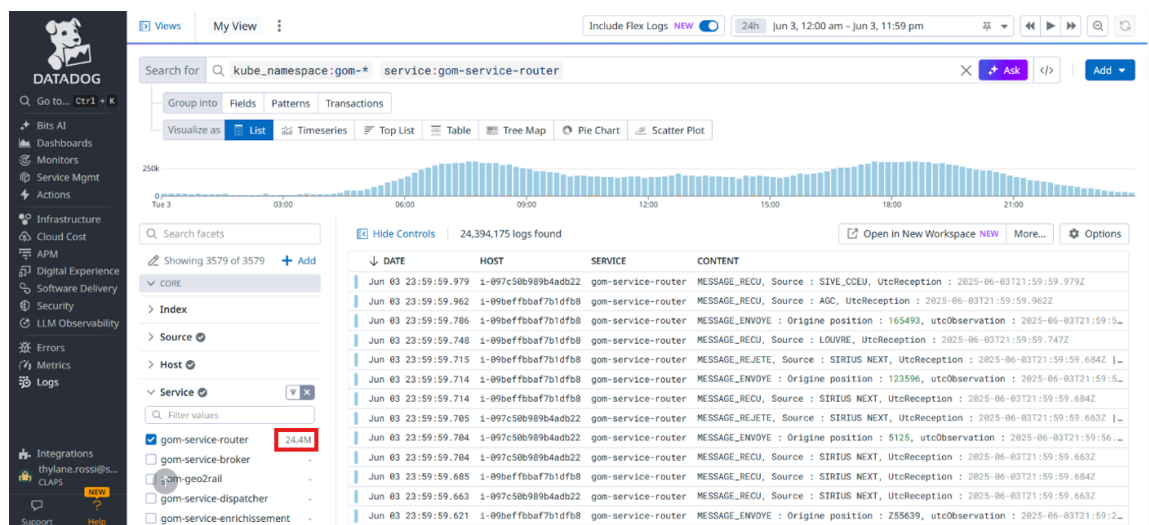


Figure 14 - Extrait des logs du deploy gom-service-router sur 24h

Comme dit plus haut, ma mission principale est de créer une interface qui facilite la gestion des logs envoyés à Datadog dans le but de limiter les coûts économiques et environnementaux de l'indexation et du stockage de ces logs. Plusieurs paramètres peuvent être modifiés et avoir un impact significatif sur le nombre de logs envoyés, comme la rétention, ou encore le type de logs envoyé (INFO, ERROR, WARNING...). La figure 15 illustre la différence de coût sur un mois entre une rétention des logs de 7 jours, estimée à 14 707 €, et une rétention de 30 jours, dont le coût s'élève à 26 439,89 €.

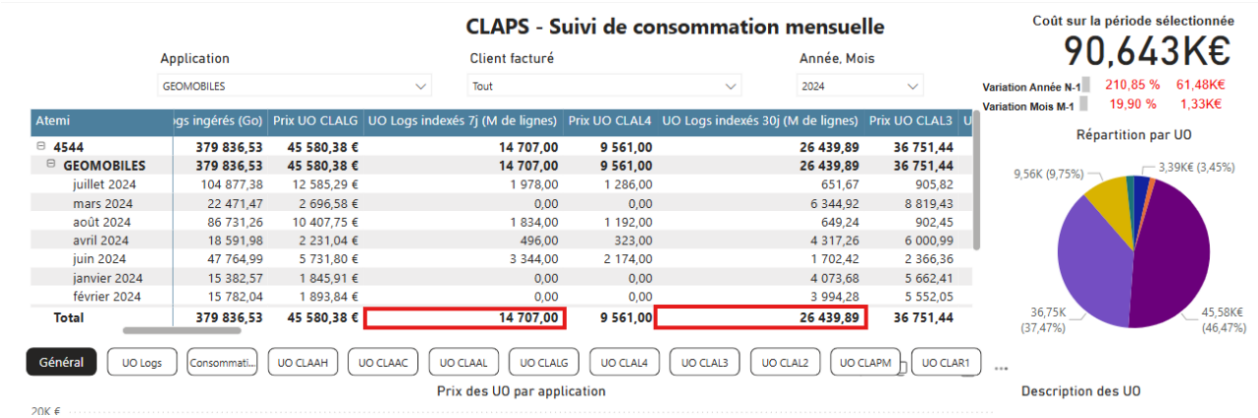


Figure 15 - Tableau de suivi de la consommation mensuelle en fonction de la rétention

Encore un exemple du prix des logs, la figure 16 ci-dessous met en évidence l’impact des durées de rétention (3, 7, 15 jours, etc.) sur les coûts, soulignant encore une fois l’importance d’appliquer des filtres adaptés pour limiter l’envoi de logs inutiles et optimiser les ressources.

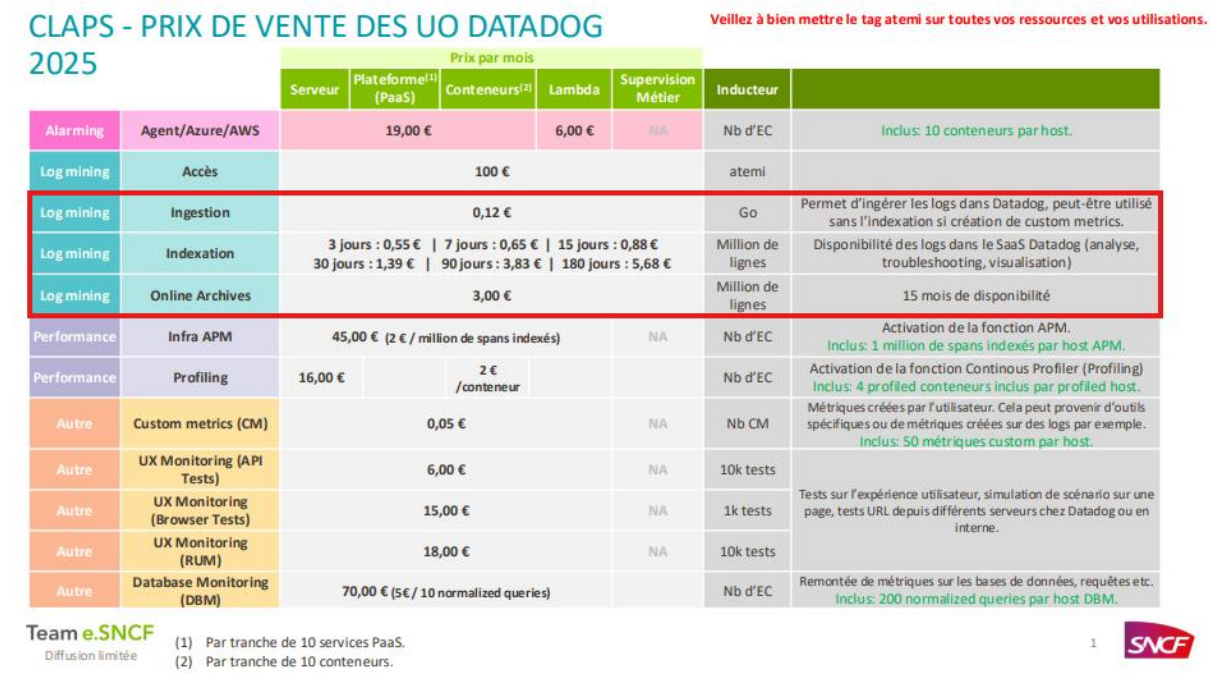


Figure 16 - Prix de vente des UO Datadog 2025

## LE TRAVAIL REALISE

Pour toutes ces raisons, le travail qui m'a été confié prend tout son sens. Face à la quantité importante de logs générés et aux coûts associés, il devenait nécessaire de mettre en place un outil permettant de mieux visualiser, filtrer et ajuster leur collecte. C'est dans ce cadre que s'inscrit ma mission, que je détaille dans la partie suivante.

## L'AFFICHAGE GENERAL DES ENVIRONNEMENTS

L'une de mes premières missions a été de concevoir l'interface d'accueil permettant d'afficher, pour chaque environnement, son namespace Kubernetes, les ressources déployées, ainsi que les informations de rétention de logs associées. Cette interface se présente sous forme de cartes dynamiques, filtrables par application, offrant une vision synthétique et lisible des données.

Techniquement, la donnée affichée sur cette page provient d'un traitement backend centralisé dans le fichier `services/datadog.py`. Plus précisément, une fonction nommée `enrich_logs()` s'exécute en arrière-plan dès le démarrage de l'application. Elle interroge l'API Kubernetes pour parcourir les déploiements (Deployments et StatefulSets) dans chaque namespace configuré, en extrayant les annotations relatives à Datadog (tags, règles de traitement de logs, etc.). Ces données sont enrichies, regroupées et stockées dans une variable globale partagée `shared_data.logs_enriched`.

Côté Flask, une route API (`/api/datadog-enriched`) exposée dans `routes/datadog.py` permet de récupérer ce JSON enrichi via une requête HTTP GET. Lorsque l'utilisateur arrive sur la page `/datadog/`, le HTML `templates/datadog.html` est affiché et un script JavaScript (`static/datadog.js`) se charge d'appeler cette API pour injecter dynamiquement les cartes dans la page. Le JavaScript récupère le JSON envoyé par l'API et crée une carte pour chaque environnement.

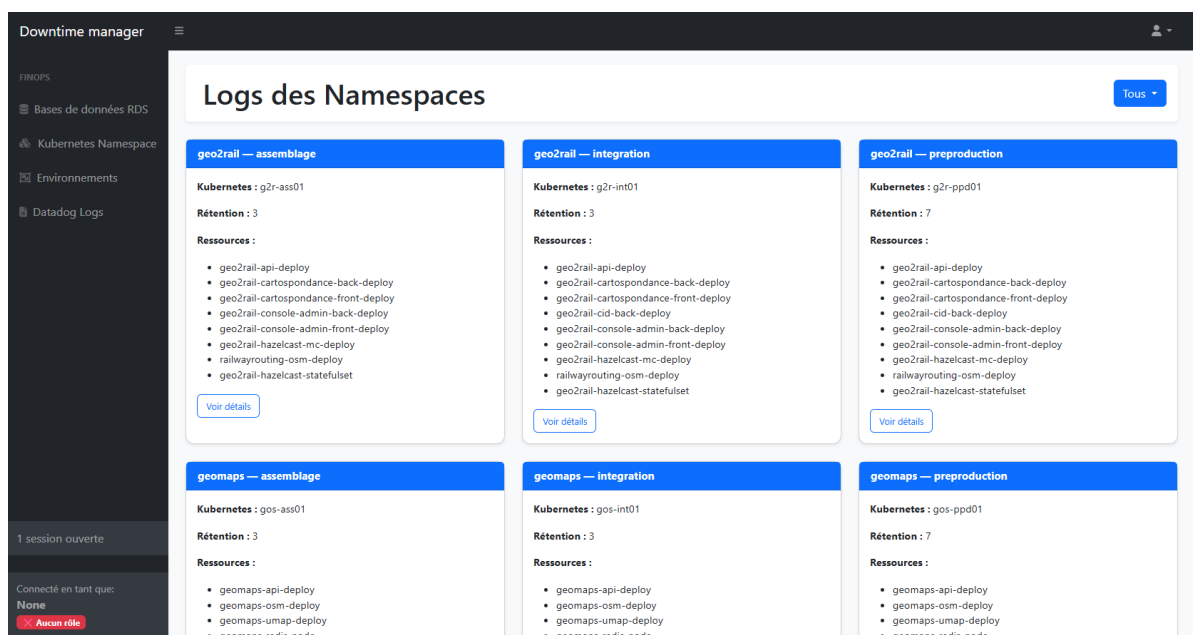


Figure 17 - Interface d'affichage des environnements

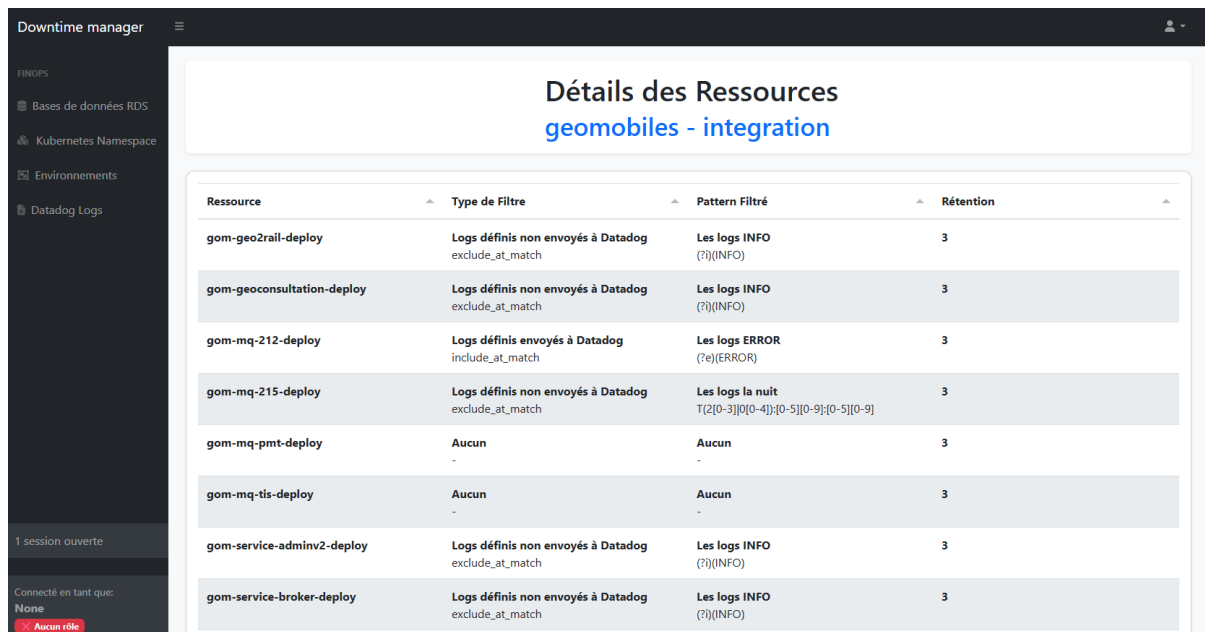
## L’AFFICHAGE DES DETAILS DE CHAQUE RESSOURCE

Ma deuxième mission a été la conception et le développement d’une interface web permettant d’afficher les détails des ressources d’un environnement donné, en s’appuyant sur les données issues de la plateforme Datadog. Cette IHM a été développée en intégrant le framework Flask côté serveur, et Tabulator côté client pour l’affichage interactif sous forme de tableau.

Concrètement, lorsque l’utilisateur clique sur le bouton « *Voir détails* » d’un environnement sur la première interface, une page dédiée est générée dynamiquement via le template `datadog_details.html`. Cette page contient un conteneur HTML prévu pour accueillir un tableau qui sera alimenté dynamiquement avec les données enrichies.

L’affichage repose principalement sur le fichier JavaScript `datadog_details.js`, qui est exécuté au chargement de la page. Ce script commence par extraire le nom de l’application et de l’environnement à partir des balises HTML cachées, puis envoie une requête HTTP vers l’endpoint `/api/datadog-enriched`. Ce dernier retourne l’ensemble des données enrichies, préalablement calculées côté serveur via la fonction `enrich_logs`.

Une fois les données récupérées, le script JavaScript extrait les ressources associées à l’environnement concerné. Chaque ressource comprend plusieurs informations, comme le nom, le type de filtre appliqué, le pattern de filtrage, ainsi que la durée de rétention des logs. Ces données sont ensuite formatées en lignes, et passées à la librairie Tabulator, qui se charge de construire dynamiquement le tableau HTML.



Ressource	Type de Filtre	Pattern Filtré	Rétention
gom-geo2rail-deploy	Logs définis non envoyés à Datadog exclude_at_match	Les logs INFO (?)(INFO)	3
gom-geoconsultation-deploy	Logs définis non envoyés à Datadog exclude_at_match	Les logs INFO (?)(INFO)	3
gom-mq-212-deploy	Logs définis envoyés à Datadog include_at_match	Les logs ERROR (?e)(ERROR)	3
gom-mq-215-deploy	Logs définis non envoyés à Datadog exclude_at_match	Les logs la nuit T(2[0-3][00-4])[0-5][0-9]:[0-5][0-9]	3
gom-mq-pmt-deploy	Aucun -	Aucun -	3
gom-mq-tis-deploy	Aucun -	Aucun -	3
gom-service-adminv2-deploy	Logs définis non envoyés à Datadog exclude_at_match	Les logs INFO (?)(INFO)	3
gom-service-broker-deploy	Logs définis non envoyés à Datadog exclude_at_match	Les logs INFO (?)(INFO)	3

Figure 18 - Interface du détail de chaque ressource d’un environnement (ici Geomobiles – intégration)

## OPTIMISATION DU TEMPS DE CHARGEMENT DES DONNEES

Ma troisième mission a consisté à optimiser le chargement initial des données de l'application, en mettant en place un système de préchargement en parallèle à l'aide de threads Python. L'objectif était de réduire le temps d'attente des utilisateurs.

La fonction centrale de ce système est `preload_all_data()` (voir Figure 19), située dans le fichier `app.py`. Elle est appelée automatiquement au démarrage de l'application Flask, et lance plusieurs threads en parallèle pour charger différentes sources de données :

- les logs Datadog,
- les bases de données RDS,
- et les annotations Kubernetes.

```
def preload_all_data(flask_app):
    # Cette fonction lance des threads en parallèle pour charger datadog // RDS // Annotations // Puis exécute enrich_applications_with_metadata
    try:
        with flask_app.app_context():
            flask_app.logger.info("Démarrage du préchargement des données...")

            if not shared_data.logs_enriched: # Traitement du préchargement des données datadog
                global thread_logs
                thread_logs = threading.Thread(target=enrich_logs_thread, args=(shared_data,)) # Le thread pour enrichir les logs
                thread_logs.start() # Lancement du thread
                builtins.thread_logs = thread_logs

            if not shared_data.rds_states: # Traitement du préchargement des RDS (en parallèle dans load_rds_info)
                global thread_rds
                thread_rds = threading.Thread(target=load_rds_info_thread, args=(flask_app, shared_data, True)) # Le thread pour enrichir les rds
                thread_rds.start() # Lancement du thread
                builtins.thread_rds = thread_rds
                thread_rds.join()

            # Traitement du préchargement des annotations
            global thread_annotations
            thread_annotations = threading.Thread(target=get_namespace_annotations_thread, args=(flask_app, annotations,))
            thread_annotations.start() # Lancement du thread
            builtins.thread_annotations = thread_annotations
            thread_annotations.join()

            if not shared_data.applications_enriched:
                # Attendre que rds_states et annotations soient rempli pour exécuter enrich_application_with_metadata
                thread_rds.join()
                thread_annotations.join()

                shared_data.applications_enriched = enrich_applications_with_metadata(
                    copy.deepcopy(shared_data.applications),
                    annotations,
                    shared_data.rds_states
                )
                flask_app.logger.info("✅ applications_enriched chargé")

    except Exception as e:
        flask_app.logger.exception("❌ Erreur dans preload_all_data")
```

Figure 19 - Fonction `preload_all_data()` du fichier `app.py` qui lance le chargement des données en parallèle

La partie la plus complexe concerne le chargement des logs. La fonction `enrich_logs()`, appelée dans un thread dédié, gère deux niveaux de parallélisme :

- un premier niveau de threads pour traiter chaque application indépendamment (voir Figure 20 : les différentes applications Géoservices se lancent même si celle d'avant n'est pas finie),
- puis un second niveau pour traiter chaque namespace Kubernetes associé à ses environnements (voir Figure 20 : les différents threads des namespaces qui se terminent).



Chaque sous-thread va interroger les ressources du cluster Kubernetes et extraire les annotations liées aux logs (filtres, règles de traitement, politique de rétention, etc.). Les résultats sont ensuite centralisés de manière sécurisée à l'aide d'un verrou (threading.Lock) pour éviter les conflits entre threads.

Grâce à cette architecture, le temps de traitement a été réduit de façon significative. Par exemple, le chargement de l'interface des logs qui prenait plus de 13 secondes ne prend désormais que 2.83 secondes, comme le montre une capture d'écran issue de mes tests. Ce gain de performance a un impact direct sur la fluidité et la réactivité de l'interface utilisateur.

```
(venv) C:\B341586\W00530P:~/OK/downtime-manager/downtime-manager$ flask run
[06/10/25 15:14:23] INFO Recovering applications data from external source!
* Debug mode: off
INFO Démarrage du préchargement des données...
DEBUG Démarrage de geomobiles
DEBUG Démarrage de geomaps
DEBUG Démarrage de geonotif
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
DEBUG Démarrage de geo2rail
DEBUG Démarrage de geopulse
DEBUG Démarrage de gpstrains
Démarrage de geomobiles
Démarrage de geomaps
Démarrage de geonotif
Démarrage de geo2rail
Démarrage de geopulse
Démarrage de gpstrains
Fin de gpstrains en 0.8s
Fin de g0i-ass01 en 0.26s
Fin de gos-ass01 en 0.3s
Fin de gnf-ass01 en 0.57s
Fin de g2r-ass01 en 0.66s
Fin de gos-int01 en 0.46s
Fin de g0i-int01 en 0.49s
Fin de gos-ass01 en 0.94s
Fin de gnf-int01 en 0.43s
Fin de g0i-rec01 en 0.33s
Fin de gos-ppd01 en 0.5s
Fin de geomaps en 1.26s
Fin de g2r-int01 en 0.67s
Fin de g0i-ppd01 en 0.37s
Fin de geopulse en 1.46s
Fin de gnf-rec01 en 0.56s
Fin de g2r-ppd01 en 0.32s
Fin de geo2rail en 1.66s
Fin de gos-int01 en 0.7s
Fin de gnf-ppd01 en 0.69s
Fin de geonotif en 1.68s
[06/10/25 15:14:25] INFO Lancement du thread annotations
Fin de gos-rec01 en 0.23s
Fin de gos-rec02 en 0.1s
Fin de gos-ppd01 en 0.22s
Fin de geomobiles en 2.22s
enrich_logs terminé en 2.83
[06/10/25 15:14:25] INFO applications_enriched chargé
127.0.0.1 - - [10/Jun/2025 15:14:27] "GET /socket.io/?EIO=4&transport=polling&t=jltlbjhz HTTP/1.1" 200 -
127.0.0.1 - - [10/Jun/2025 15:14:27] "POST /socket.io/?EIO=4&transport=polling&t=jltlbjhz&sid=oZrOVY6VkeVqY-ZtAAAA HTTP/1.1" 200 -
127.0.0.1 - - [10/Jun/2025 15:14:27] "GET /socket.io/?EIO=4&transport=polling&t=jltlbjhz&sid=oZrOVY6VkeVqY-ZtAAAA HTTP/1.1" 200 -
```

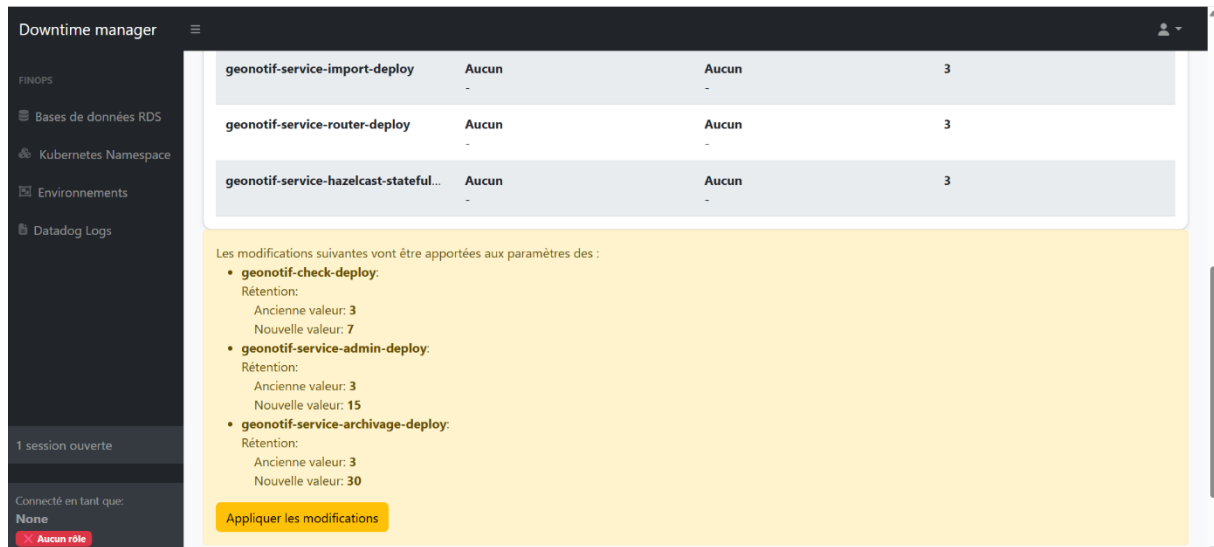
Figure 20 - Extrait du terminal lors du chargement des logs

## MODIFICATION DE LA RETENTION

Une des tâches techniques qui m'a ensuite été confiée a été d'ajouter la possibilité de modifier la rétention des logs pour une ressource Kubernetes (comme un Deployment ou un StatefulSet). Cette information est stockée dans les annotations Kubernetes, utilisées par Datadog pour la durée de conservation des logs.

Concrètement, j'ai travaillé sur une interface en JavaScript qui liste toutes les ressources par namespace. Chaque ligne affiche les règles de filtrage des logs ainsi que la rétention associée. Grâce à la bibliothèque Tabulator, l'utilisateur peut modifier directement la rétention dans le tableau via un menu déroulant. Dès qu'un champ est modifié, la modification est sauvegardée localement et affichée dans une section récapitulative (voir Figure 21).





Downtime manager			
geonotif-service-import-deploy	Aucun	Aucun	3
geonotif-service-router-deploy	Aucun	Aucun	3
geonotif-service-hazelcast-stateful...	Aucun	Aucun	3

Les modifications suivantes vont être apportées aux paramètres des :

- geonotif-check-deploy:**
  - Rétention:
    - Ancienne valeur: 3
    - Nouvelle valeur: 7
- geonotif-service-admin-deploy:**
  - Rétention:
    - Ancienne valeur: 3
    - Nouvelle valeur: 15
- geonotif-service-archivage-deploy:**
  - Rétention:
    - Ancienne valeur: 3
    - Nouvelle valeur: 30

Appliquer les modifications

Figure 21 - Résumé des modifications des rétentions des déploiements

Une fois les changements validés, ils sont envoyés au backend via WebSockets (voir Figure 22). Côté serveur, une fonction Flask réceptionne les modifications, identifie la ressource concernée et applique les nouvelles annotations Kubernetes via l'API Python. Si la ressource n'existe pas ou qu'une erreur est rencontrée, un message d'erreur est renvoyé pour ne pas bloquer l'utilisateur.

Enfin, après chaque mise à jour, un rafraîchissement global est déclenché pour recharger les données et s'assurer que l'interface reflète bien les nouvelles rétentions.

```
@socketio.on("update_logs_tag")
@login_required_oidc_socketio
def update_logs_tag(data):
    username = session.get('username', 'anonymous')
    user_session = f"{username}({request.sid})"
    app.logger.info(f"✳️ {user_session} demande une mise à jour des logs")
    app.logger.debug(f"📁 Données reçues : {data}")

    responses = {}

    for app_name, envs in data.items():
        for env_name, resources in envs.items():
            env_data = shared_data.logs_enriched.get(app_name, {}).get(env_name, {})
            if not env_data:
                app.logger.warning(f"⚠️ Données introuvables pour {app_name}/{env_name}")
                continue

            for resource_name, changes in resources.items():
                namespace_found = None
                for ns, logs in env_data["resources"].items():
                    if any(entry["resource_name"] == resource_name for entry in logs):
                        namespace_found = ns
                        break

                if not namespace_found:
                    app.logger.warning(f"⚠️ Namespace introuvable pour {resource_name}, ignoré.")
                    responses[resource_name] = {"status": "error", "message": "Namespace introuvable"}
                    continue

                annotations = {}

                # Rétention
                if "log_retention" in changes:
                    annotations["ad.datadoghq.com/tags"] = json.dumps({
                        "log-retention": changes["log_retention"]["newTagValue"]
                    })
```

Figure 22 - Bout de code de la socket pour la modification de la rétention

## LA SUITE DU STAGE

Pour la suite du stage, qui se déroule jusqu'au 8 août, il me reste encore des tâches à réaliser, qui sont détaillées juste après.

---

### MODIFICATION DES LOGS ENVOYES A DATADOG

Après avoir mis en place la modification de la rétention des logs, je travaille désormais sur une nouvelle fonctionnalité : le filtrage des logs envoyés à Datadog. L'objectif est de permettre aux utilisateurs de définir, pour chaque ressource, une règle d'envoi (`exclude_at_match` ou `include_at_match`) ainsi qu'un *pattern* associé, c'est-à-dire une expression régulière décrivant les logs à filtrer. Par exemple, `(?i)(INFO)` permet de cibler uniquement les logs de type INFO, tandis qu'un pattern comme `T(2[0-3][0[0-4]]:[0-5][0-9]:[0-5][0-9])` permet d'isoler ceux émis la nuit.

Cette partie s'avère plus complexe que la gestion de la rétention. En effet, les champs `rule_type` et `pattern` sont liés : il n'est pas possible d'en modifier un sans modifier l'autre. De plus, le tag utilisé dans les annotations Kubernetes est différent de celui de la rétention, ce qui ajoute une difficulté technique supplémentaire. Malgré cela, le principe de modification reste globalement le même, avec une mise à jour des annotations côté backend et un rafraîchissement des données côté interface.

---

### RESTRUCTURATION DU CODE AVEC UNE CLASSE

Quand les modifications seront réussies, une des pistes d'amélioration envisagées concerne la restructuration du code afin de le rendre plus clair, plus modulaire et plus facile à maintenir. Pour cela, l'idée proposée par les OPS serait d'introduire une classe dédiée à la gestion des logs, avec un constructeur capable de centraliser les dépendances nécessaires (comme le contexte de l'application Flask).

Aujourd'hui, certaines fonctions, notamment celles présentes dans les fichiers de services, nécessitent d'accéder au contexte global de l'application. Dans les routes, cela se fait facilement avec `current_app`, mais dans les fonctions "métiers", il est préférable de passer explicitement ce contexte en paramètre, ce qui alourdit un peu le code. L'objectif de cette refonte serait donc de regrouper ces fonctions dans une même classe, et de passer le contexte une seule fois via le constructeur. Cela permettrait de limiter les imports croisés, d'améliorer la lisibilité générale, et de rendre le code plus facilement testable, notamment dans le cadre de futurs tests unitaires.

---

### TESTS UNITAIRES POUR UN DEPLOIEMENT DE LA NOUVELLE VERSION

Une fois le code restructuré de manière plus claire et plus modulaire, une étape essentielle consistera à mettre en place des tests unitaires. L'objectif est de s'assurer que chaque fonction, chaque scénario d'utilisation et chaque modification apportée se comportent

comme prévu. Ces tests permettront de valider la fiabilité de la nouvelle fonctionnalité, tout en facilitant le déploiement de la future version de Downtime Manager. Ils joueront également un rôle clé pour éviter les régressions lors des évolutions ultérieures du projet.

---

#### PRESENTATION DE L'APPLICATION AUX AUTRES SERVICES SNCF

Par ailleurs, une présentation de l'application Downtime Manager est prévue les 16 et 18 juin, avec les deux OPS, auprès d'autres équipes de la SNCF et de l'évènement Kube@Lyon. L'objectif est de faire connaître l'outil et d'envisager son déploiement au sein d'autres équipes, comme cela a déjà été réalisé avec succès dans un autre service. Cela montre que le projet commence à susciter de l'intérêt en dehors de l'équipe d'origine, ce qui ouvre des perspectives concrètes de valorisation à plus grande échelle.

#### CONCLUSION

Même si mon stage n'est pas encore terminé, je peux déjà tirer un bilan très positif de cette expérience. J'ai eu l'opportunité de m'investir sur un vrai projet métier, avec des enjeux concrets à la fois techniques, financiers et environnementaux. Le développement de la nouvelle interface de gestion des logs dans Downtime Manager m'a permis de mobiliser des compétences acquises pendant ma formation, tout en en développant de nouvelles, notamment autour de Flask, Kubernetes, ou encore Datadog.

Au-delà de l'aspect technique, ce stage m'a également permis de mieux comprendre les méthodes de travail d'une grande entreprise, l'importance de la collaboration, des revues de code et du suivi de projet. L'accompagnement des OPS a été particulièrement formateur et bienveillant, ce qui a largement contribué à la qualité de cette expérience.

La suite du stage s'annonce tout aussi stimulante, avec la finalisation de la fonctionnalité de filtrage, la restructuration du code et la mise en place de tests unitaires. Par ailleurs, la présentation de l'outil à d'autres équipes de la SNCF témoigne de l'intérêt qu'il suscite, et laisse entrevoir de belles perspectives pour la suite du projet.

## ANNEXE

Annexe 1 - Documentation Datadog pour comprendre les filtres :

[https://docs.datadoghq.com/fr/agent/logs/advanced\\_log\\_collection/?tab=fichierdeconfiguration](https://docs.datadoghq.com/fr/agent/logs/advanced_log_collection/?tab=fichierdeconfiguration)

Annexe 2 - Vidéo qui représente la densité de train sur 24h :

<https://x.com/SNCFNumerique/status/1579488104364531713>

Annexe 3 – Les domaines d’expertises de e.SNCF, je travaille dans hébergement & exploitation applicative :



### RAISON D’ÊTRE

Nous sommes la fabrique numérique du Groupe SNCF.

### MISSIONS

- Être au service des DSI et des métiers
- Assurer la sécurité du SI du Groupe SNCF
- Produire un numérique plus responsable et plus inclusif
- Maintenir un haut niveau de qualité de nos services

## NOS DOMAINES D’EXPERTISE



Nos domaines d’expertise intégrant déjà l’IA.



**CYBERSÉCURITÉ** Sécurisation des SI



**APPLICATIONS DES FONCTIONS SUPPORT**

DSI des fonctions support :  
Finances, RH, Achats,  
Communication...  
& Fab AR-VR



**DIGITAL WORKPLACE**

Bureautique, téléphonie et  
services métiers (logiciels,  
signature électronique...)



**CONNECTIVITÉS**

Gestion des connectivités  
entre les équipements et les  
plateformes de services  
(wifi, accès au SI...)



**DESIGN & DÉVELOPPEMENT D’APPLICATIONS**

Conception et développement  
d’applications  
& Laboratoire d’innovations  
et de Prototypes



**BUILD**



**HÉBERGEMENT & EXPLOITATION APPLICATIVE**

Gestion des serveurs  
internes ou Cloud et de  
l’exploitation des  
applications



**& RUN**



**ÉCOLE NUMÉRIQUE**

Au service des  
collaborateurs de l’IT mais  
également de tous les  
agents SNCF.



**SOURCING LOGICIELS**

Gestion des relations  
contractuelles avec les  
éditeurs de logiciels