Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE de fin de deuxième année
Bachelor Universitaire de Technologie
Spécialité Réseaux et Télécommunications
parcours cybersécurité

Automatisation des sanity check

Sebastian DESMEDT

AIRFRANCE KLM

Responsable en entreprise : Romain SERCER

Responsable académique : Delphine ROUSSEAU

2023

Table des matières

1. Introduction

À l'intention du lecteur : Le caractère international du groupe Air France KLM, nous oblige à de nombreux anglicismes. Veuillez donc ne pas vous formaliser face à cette alternance linguistique.

1.1. Présentation d'AIRFRANCE KLM

Le groupe Air France-KLM[1], né en 2004 de la fusion entre Air France et KLM, est le premier groupe de transport aérien en Europe et un leader mondial. Outre le transport de passagers, qui constitue 80 % de son chiffre d'affaires, le groupe est un acteur majeur du transport de fret avec AFKL Cargo (11 % du chiffre d'affaires). Il propose aussi des services de maintenance aéronautique via AFKL E&M (4,5 % du chiffre d'affaires) et des solutions informatiques avec AFKL IT Solutions (5,5 % du chiffre d'affaires).

Compagnies aériennes premium, Air France et KLM mettent ses clients au cœur de ses priorités en fournissant des prestations de grande qualité, qui se reflètent dans des coûts privilégiés.
Pour contrevenir à cette clientèle exclusive, Air France-KLM propose, par le biais de sa filiale Transavia, des vols charters à des tarifs compétitifs accessibles à tous les voyageurs.

Des professionnels de l'aéronautique aux autres compagnies aériennes, le groupe bénéficie d'une clientèle diverse grâce à ses activités variées, dont l'essentiel est le voyageur d'affaires.

1.2. Culture d'Entreprise

Air France valorise l'inclusion et la collaboration, convaincue que la diversité enrichit l'innovation. Elle favorise l'équilibre vie professionnelle et vie privée avec des options flexibles comme le télétravail (jusqu'à 4 jours par semaine pour certains postes) et encourage l'évolution et la mobilité interne.
La santé et le bien-être des employés sont primordiaux, avec un cadre de travail ouvert (open space)[2] encourageant la communication et la collaboration pour atteindre des objectifs communs.

1.3. Présentation du service

Air France m'a accueilli au sein de la direction Distributed Services[3], dans l'équipe Datacenter Network dirigée par M. Christophe Bigot, au sein du service des Télécoms dirigé par M. Amine Benkirane (*cf. figure 1 organigramme)*. Cette équipe est composée de deux sous-équipes : ITDS TZ[4], dirigée par M. Stéphane Dolique, et ITDS TD, également sous la direction de M. Christophe Bigot. Réparties sur quatre sites (*cf. figure 2 schéma du service),*

[1] Abrégé par AFKL
[2] Bureau ouvert, sans porte ni cloison
[3] Se traduit par Services Distribués.
[4] Cet acronyme signifie IT Distributed Services Team Z qui se traduit par équipe Z des services distribués.

ces sous-équipes se divisent en quatre « équipes virtuelles » afin de dissocier clairement les activités de chacun[5].

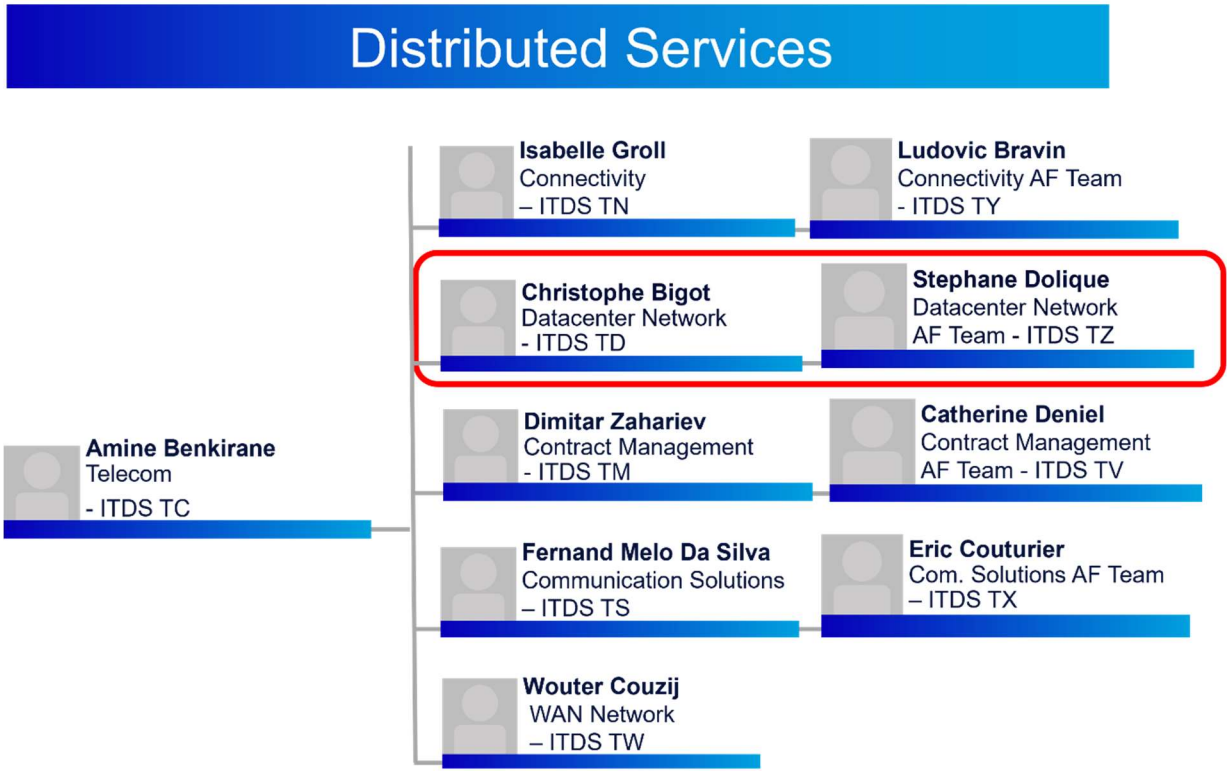

Figure 1 organigramme

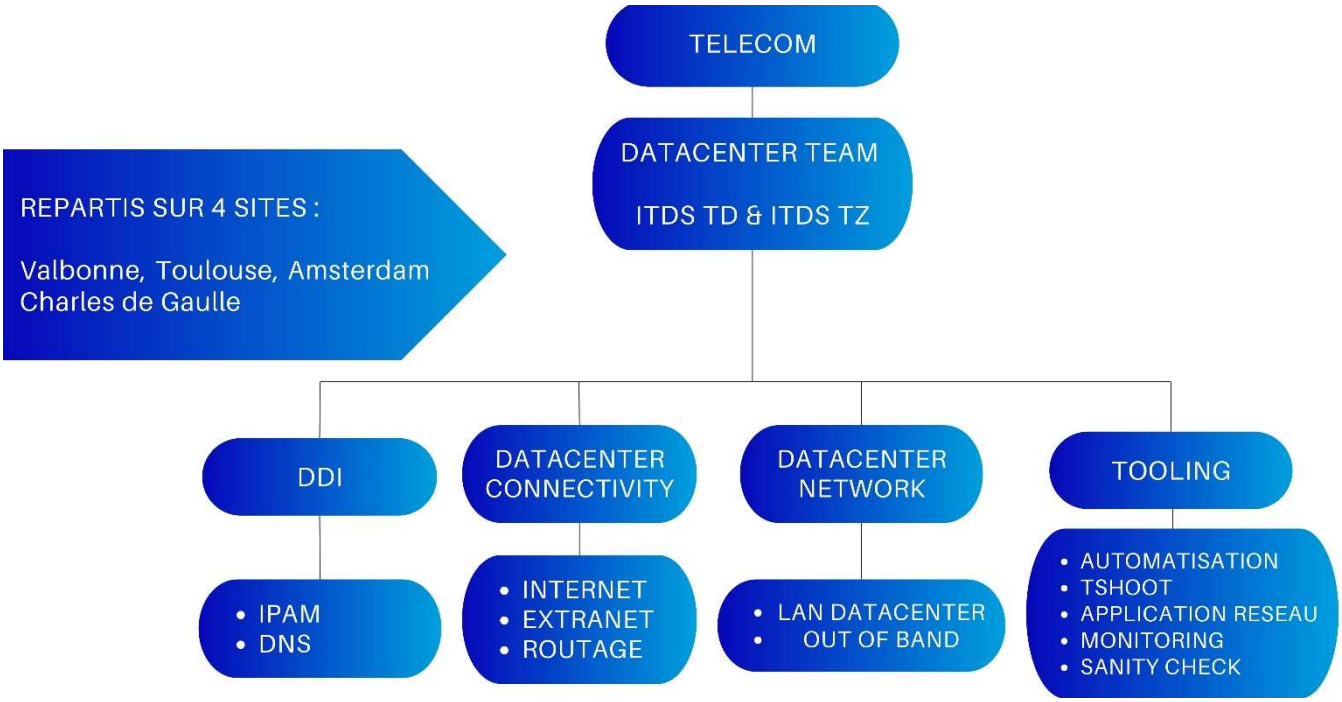

Figure 2 schéma de service

Le dernier étage représente de façon non exhaustive, les activités de chaque équipe.

[5] Au vu de la technicité des tâches présentées dans le schéma du service, nous faisons grâce au lecteur d'explications laborieuses susceptibles de le désorienter.

1.4. Cadre technique général du sujet

Les sanity checks, essentiels au processus de gestion de la continuité[6] chez Air France, sont des vérifications rudimentaires destinées à assurer le bon fonctionnement des systèmes et l'intégrité des données. Hebdomadaire ou journalier ; ils sont accomplis à la main. Cette pratique bien que chronophage, est vitale pour prévenir les incidents de production. L'état actuel des choses fait que les équipes NetOps[7] reçoivent un flot d'alarmes[8] difficilement gérable. Cette impéritie peut entraîner des failles ou des erreurs qui restent ignorées ou tardivement résolues. L'automatisation des sanity check permettra de traiter une majorité d'alarmes, réduisant ainsi la charge des équipes préposées.

Les Jeux Olympiques approchent, et avec eux de nombreux voyageurs à destination de la France, qui sont autant de trafics pour les infrastructures réseau, mais également l'enjeu de devoir représenter la France sur la scène internationale. L'automatisation des sanity check est donc un impératif pour aborder cette période d'affluence avec sérénité.

1.5. Cahier des charges

Durant les 10 semaines de stage, les missions suivantes ont été accomplies :

- Comprendre l'infrastructure réseau du groupe ainsi que les différents concepts sur lesquels elle est bâtie.
- Comprendre les différentes responsabilités des équipes et du département Télécom.
- Développement de scripts en Python et JavaScript pour automatiser les processus internes.
- Optimisation et administration d'outils internes pour améliorer l'efficacité opérationnelle.
- Participation active aux réunions du service pour discuter des avancements et présenter les résultats obtenus.

[6] Le processus de gestion de la continuité désigne l'ensemble des mesures et procédures mises en place pour assurer le maintien et la reprise rapide des fonctions essentielles à la suite d'une perturbation majeure.
[7] NetOps, Network Operations (opérations réseau), désigne les activités et procédures dédiées à la gestion, à la surveillance et au maintien du fonctionnement optimal des réseaux informatiques.
[8] La gestion des alarmes fait partie intégrante des sanity checks.

1.6. Organisation des projets et du temps de travail

Au cours de mon stage, j'ai travaillé sur deux projets distincts (*cf. tableau 1 planning des projets)* ; il a donc été nécessaire de faire preuve de rigueur et d'organisation. Le planning suivant vise à apporter une meilleure compréhension du déroulement de mon stage et des différentes phases de chaque projet.

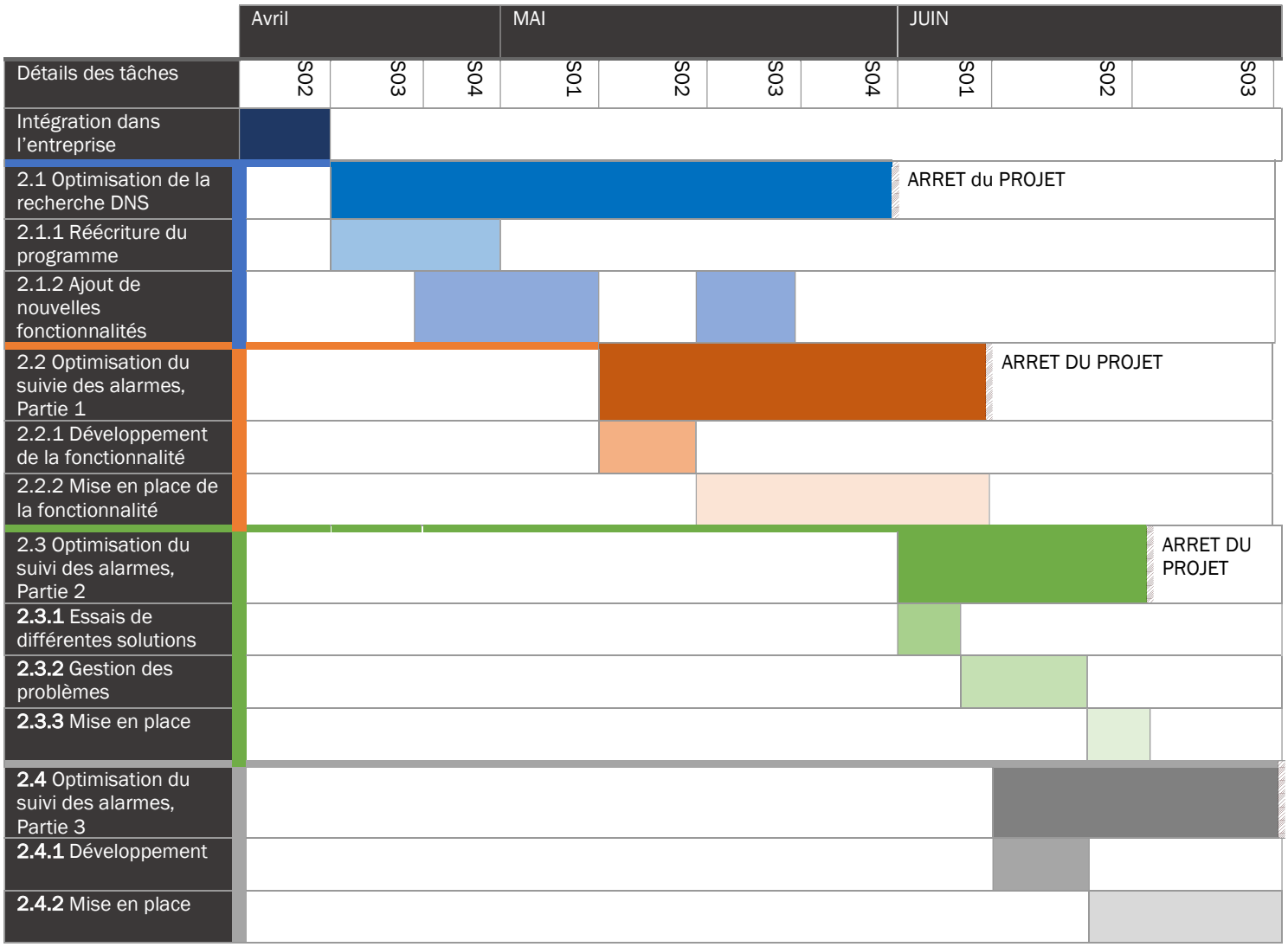

Tableau 1 planning des projets

2. Travails réalisés

2.1 Optimisation et amélioration de la récupération d'information DNS

2.1.1 Présentation préalable des outils

Avant d'aborder ce projet, il est essentiel de comprendre les deux principaux outils utilisés.

1. Infoblox :

Infoblox est une solution de gestion des adresses IP (IPAM[1]) et des services DNS[2]. Elle permet aux utilisateurs de consulter et gérer les configurations via une interface utilisateur intuitive ou une API[3]. Infoblox utilise le concept de « network container[4] » pour organiser et administrer ces adresses IP de manière efficace. Cette approche est essentielle pour optimiser la gestion des réseaux dans des environnements complexes, un point central pour la suite de notre rapport. Ce principe est illustré par la figure suivante (*cf. figure 3 Principe du « network container »)*

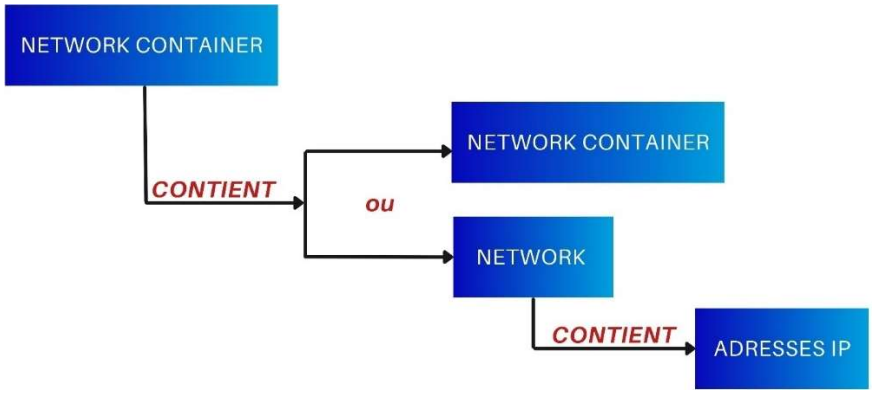

Figure 3 Principe du "network container"

2. Red Hat Ansible Automation Platform (anciennement Tower) :
   (Il sera nommé « Tower » tout au long de ce rapport)

Ansible Tower est un outil d'automatisation, de gestion et d'exécution de scripts Ansible offrant une interface utilisateur et une API. Il permet l'industrialisation des outils, orchestrant un ensemble de tâches, et est notamment utilisé pour interagir avec Infoblox.

[1] IPAM (IP Address Management) **:** Il s'agit de la gestion des adresses IP, qui comprend la planification, le suivi et la gestion de l'espace d'adressage IP au sein d'un réseau. L'IPAM est crucial pour éviter les conflits d'adresses et assurer une utilisation efficace des adresses IP disponibles.
[2] DNS (Domain Name System) **:** Gère la résolution des noms de domaine, comme airfranceklm.com, en adresses IP correspondantes, facilitant ainsi l'accès aux sites internet à travers des noms faciles à retenir au lieu de numéros complexes.
[3] Défini et détaillé plus en aval.
[4] Se traduit par, Conteneur de réseau.

2.1.2 Contexte et objectifs

L'analyse et la récupération d'informations DNS sont nécessaires au service des télécoms (*cf. figure 1 Organigramme*) pour la gestion de l'IPAM. Ce script, nommé « GET DNS HOST.py », est exécuté par un script Ansible éponyme. Sa vétusté ne correspond plus aux exigences de maintenabilité de la compagnie et nécessite donc une mise à niveau sur tous les aspects. Mon objectif était donc de réécrire le script pour le rendre plus lisible, maintenable, complet et facile à utiliser.

2.1.3 Fonctionnement du script

Le fonctionnement est illustré par le schéma suivant (*cf. figure 4 Fonctionnement DNS HOST.py*), les networks containers dont il est questions sont indiqués au sein même du programme par l'utilisateur :

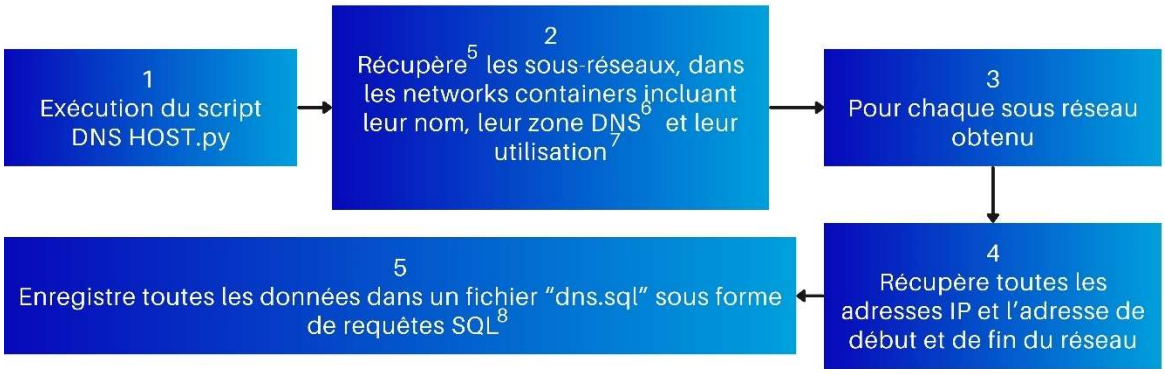

Figure 4 Fonctionnement DNS HOST.py

L'ajout à la base de données, est par la suite orchestré par le script Ansible.
La figure suivante résume l'interaction générale des différents outils et le déroulement des différentes opérations (*cf. figure 5 Interactions entre les différents outils*).

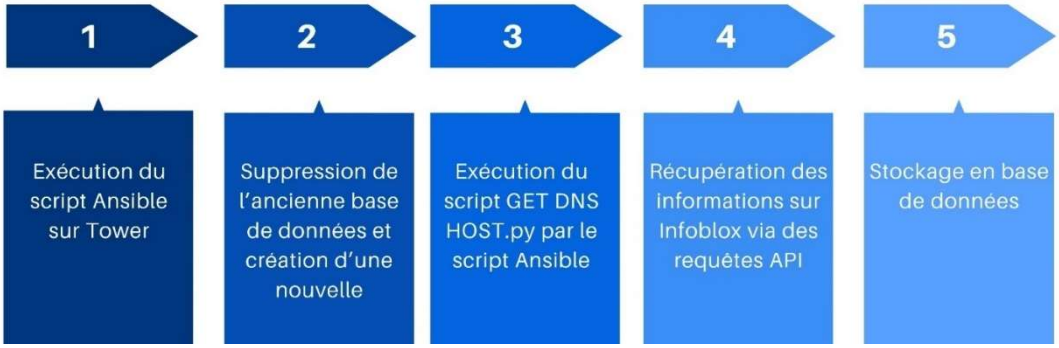

Figure 5 Interactions entre les différents outils

[5] Toute récupération d'informations auprès d'Infoblox se fait par l'intermédiaire de requêtes API.
[6] Les zones DNS sont simplement des noms qui permettent de trier les réseaux en fonction de leur zone d'utilisation, par exemple un réseau utilisé en interne sera nommé : Internal-DNS-AFKL.
[7] Pourcentage du nombre d'adresse IP qui sont affectées à une machine
[8] SQL (Structured Query Language) : Langage utilisé pour créer, gérer et interroger des bases de données. Permet de manipuler les données et d'effectuer des opérations comme l'insertion, la mise à jour, la suppression et la sélection de données.

2.1.4 Réécriture

Le respect des normes de programmation est une nécessité catégorique pour la pérennité d'un projet. Il est essentiel d'ajouter des commentaires clairs et éloquents permettant à quiconque une maintenance et une compréhension simplifiée. La factorisation du code dans des fonctions dûment définies permet de limiter les répétitions et d'optimiser l'efficacité du programme.

J'ai commencé par la factorisation du code et par la suppression de parties inutiles. Cette tâche fastidieuse a remarquablement accru la lisibilité du programme (*cf. figure 6 et 7*).

```
r = requests.get(
        url + 'network?network_container=' + networkcontainer +
        '&_return_as_object=1&_return_fields=network&_max_results=' + maximum,
        cookies=request_cookies,
        verify=valid_cert)
    duree_print("requests.get(" + url + 'network?network_container=' +
                networkcontainer +
                '&_return_as_object=1&_return_fields=network&_max_results=' +
                maximum)
    if r.status_code != requests.codes.ok:
        print(r.text)
        exit_msg = 'Error {} finding network container: {}'
        sys.exit(exit_msg.format(r.status_code, r.reason))
    results = r.json()
```

Figure 6 Exécution d'une requête avant réécriture

```
def exec_request(url:str,params:dict,cookie:dict,verify:bool=False) -> dict:

    disable_warnings(exceptions.InsecureRequestWarning) # Disable certificate absence warni
    result={"result":[]}
    request=get(
                url=url,
                params=params,
                cookies=cookie,
                verify=verify
               )

        if request.status_code != codes.ok:
            exit(f"Error {request.status_code} finding network container {request.reason}")
        request=request.json()
    return result
```

Figure 7 Exécution d'une requête après réécriture

Dans la volonté de rendre la maintenance et la compréhension du programme accessible à tous, j'ai conclu cette première partie par l'ajout de commentaires en anglais[9], jusqu'alors inexistants

[9] Nous travaillons avec des équipes situées à Amsterdam ; l'emploi de l'anglais est donc de rigueur.

(*cf. figure 8 et 9).* Chaque fonction et ses paramètres sont ainsi décrits par un commentaire similaire à celui de la *figure 8*. Ceux qui suivent (*cf. figure 9)* précisent des détails ou le fonctionnement de certaines parties ardues.

```
def get_network_container(url:str,cookie:dict,max_result:str,dnsview_ip_list:list) -> list:
    """
    Retrieves the address of all network containers and sub-containers
    Args:
        url (str): example: https://infoblox.cin/v2.11.5/
        cookie (str): cookie session generated by ansible :{'ibapauth': _cookie_}
        dnsview (list): A list of ip addresses of network container associated with a dnsview
        max_result (str): maximum number of results in a query, its value has no importance
since the introduction of pagination (but it must still be indicated).

    Returns:
        list: of all the network containers and sub-containers
    """
```

Figure 8 Commentaire ajouté sous chaque fonction

```
"""
  - Pagination control:
Pagination allows you to retrieve all the objects in a query, regardless of the number of
elements. When there are too many elements,instead of truncating the result, the query returns
a “next_page_id” field with a number. Simply define the parameters with this “next_page_id
number” to access the rest of the data until the field is no longer filled in.
"""
```

Figure 9 Commentaire détaillant le fonctionnement de la pagination

2.1.5 Recherche récursive

La première tâche qui m’a été donnée d’ajouter, fut d'élargir le champ d'investigation du script, en intégrant une recherche récursive dans les networks containers, fonctionnalité absente de notre script initial. Celui-ci, en se cantonnant aux seuls sous-réseaux (*cf. 2.2.3 fonctionnements)* négligeait les sous-conteneurs, omettant ainsi de traiter une masse significative de données, au détriment manifeste de notre information.
L’objectif était donc de parcourir les sous-conteneurs réseaux[10] et leurs réseaux associés.

Nous avons décidé de diviser le problème en deux parties distinctes :

1. La récupération des sous conteneurs réseau.

2. La récupération des réseaux contenus dans les sous conteneurs.

S’agissant de récupérer des informations depuis Infoblox notre travail s’est principalement penché sur la compréhension et la manipulation des requêtes API.

[10] Évidemment, si un sous-conteneur réseau dispose lui-même d’un conteneur réseau, nous entendons bien le parcourir, quel que soit son degré de profondeur.

Présentation préalable des requêtes API

L'interface de programmation d'application[11] est une interface visant à connecter un logiciel ou une application à d'autres systèmes distincts afin qu'ils puissent échanger leurs données. L'API se matérialise comme une passerelle d'accès à une fonctionnalité détenue par une entité indépendante (*cf. figure 10*).

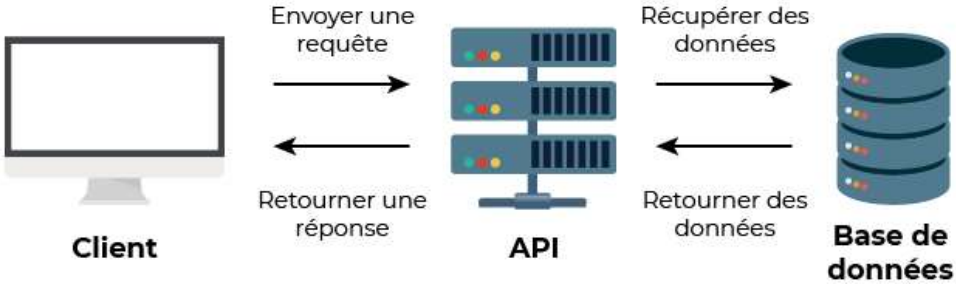

Figure 10 Interaction avec une API

Chaque requête est composée de l'url[12] de l'API, en l'occurrence « https://infoblox.cin/wapi/v2.11.5/ » à laquelle on ajoute des paramètres. Ces derniers prennent la forme de mot clés, généralement éloquents, et permettent de spécifier des objets (conteneurs réseaux, plages IP, réseaux) sur lesquels on souhaite travailler. Le tableau ci-dessous décrit les différents paramètres d'une requête API *(cf. tableau 2)* :

`https://infoblox.cin/wapi/v2.11.5/network?_return_fields=network,utilization&_max_results=10&network=192.168.1.0/24`	
Paramètres de l'URL	**Description**
« network? »	Permet de spécifier l'objet sur lequel on travaille, il s'agit donc ici d'un réseau.
« _return_fields=network, utilization »	Indique les données que l'on souhaite récupérer, l'adresse du réseau et son taux d'utilisation *cf. note 7*
« _max_results=10 »	Précise que l'on souhaite uniquement les 10 premiers résultats.
« network=192.168.1.0/24 »	Déclare l'adresse du réseau sur laquelle on souhaite travailler

Tableau 2 Description des paramètres d'une requête API

Récupération des sous-conteneurs réseaux

La requête API suivante *(cf. figure 11)* permet de récupérer tous les sous-conteneurs de premier degré dans un conteneur réseau.

```
https://infoblox.cin/wapi/v2.11.5/networkcontainer?network_container={ip}
&_paging=1&_max_results={max_result}&_return_fields=network&_return_as_object=1
```

Figure 11 Requête API, récupère tous les sous-conteneurs de premier degré

Le schéma suivant (*cf. figure 12)* illustre précisément ce qui est récupéré par la requête précédente *(cf. figure 11).*

[11] API en anglais.
[12] L'URL, pour Uniform Resource Locator, désigne communément l'adresse explicite d'un site Internet.

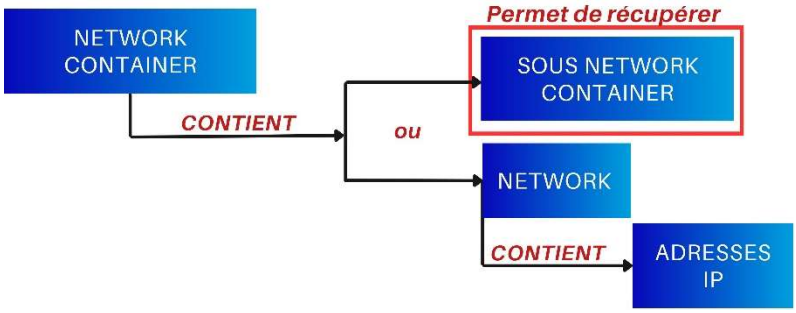

Figure 12 Sous conteneur récupéré par la requête API cf. figure 11

Les paramètres « _paging=1 », et « _return_as_object=1 » permettent respectivement, d'activer la pagination[13] et de renvoyer l'ensemble des résultats en une seule fois.
Il est donc nécessaire d'exécuter cette requête récursivement sur chaque sous-conteneur obtenu par cette même requête.

Nous avions deux possibilités pour mettre en place ce comportement, soit passer par une fonction récursive[14], soit par une boucle[15] dite, « tant que »[16]. Les fonctions récursives ont une complexité singulière qui les rend difficilement maintenables, ce qui a imposé de facto la deuxième solution. La mise en place de ce comportement est illustrée par la figure suivante *(cf. figure 13)*.

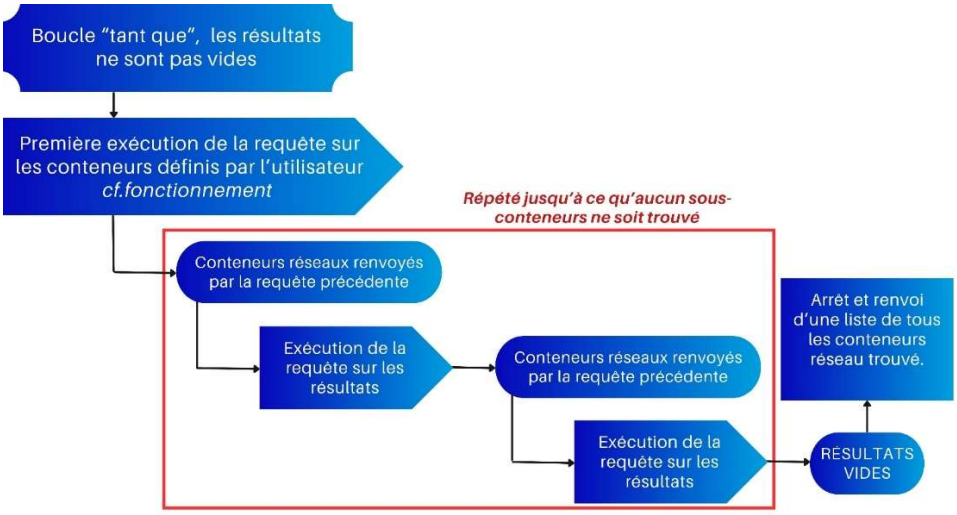

Figure 13 Récupération de tous les sous conteneurs dans un network container

Cette méthode nous permet ainsi de récupérer tous les sous-conteneurs réseau, quel que soit leur degré de profondeur.

[13] Ceci sera longuement abordé en aval
[14] Fonction qui s'appelle elle-même.
[15] Bloc d'instructions s'exécutant suivant une condition, ou selon un nombre de fois défini.
[16] Boucle qui s'exécute tant qu'une condition est respectée.

Récupération des réseaux dans les sous-conteneurs

La requête suivante permet la récupération de tous les réseaux dans un conteneur (*cf. figure 14*).

```
https://infoblox.cin/wapi/v2.11.5/network?network_container={ip}
&_return_as_object=1&_paging=1&_return_fields=network&_max_results={max_result}
```

Figure 14 Récupère les réseaux contenus dans un network container

Le schéma suivant (*cf. figure 15)* illustre précisément ce qui est récupéré par la requête précédente *(cf. figure 14).*

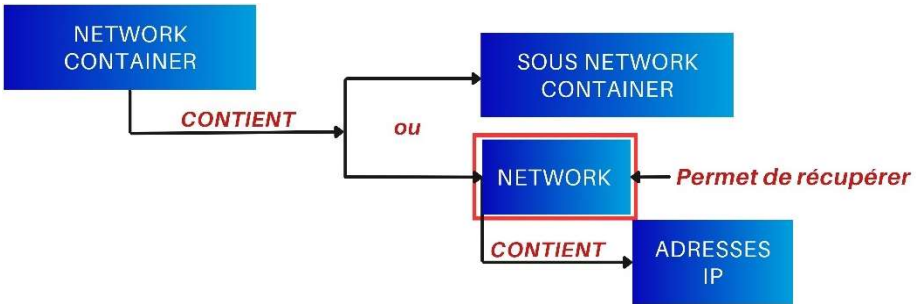

Figure 15 Réseau récupéré par la requête API cf. figure 14

Puisque la partie précédente (cf. figure 13) a permis de renvoyer une liste de conteneurs, la mise en place s'est vue grandement simplifiée. Il nous suffit donc de parcourir cette liste via une simple boucle. Le fonctionnement est illustré par la figure ci-dessous (*cf. figure 16*).

Figure 16 Récupération des réseaux

2.1.6 Mise en place de la pagination

La quantité de données restituées par les requêtes API peut être si importante que celles-ci deviennent tronquées, au détriment significatif de leur intégrité. La pagination nous permet de remédier à cela en renvoyant les données sur plusieurs requêtes, plutôt que de les amputer.

Fonctionnement et Mise en place

En premier lieu, elle doit être « activée » en précisant dans la requête API (*cf. figure 17*) les paramètres suivants : « _paging=1 », qui spécifie que l'on souhaite la pagination, et « _max_results=10 », qui indique le nombre d'éléments renvoyés par page.

```
https://infoblox.cin/wapi/v2.11.5/network?_paging=1&_max_results=10
```

Figure 17 Exemple de requête API avec la pagination

Une fois la pagination mise en place, si l'API ne peut pas renvoyer la totalité des données en une seule requête, elle ne tronquera pas les données. À la place, elle ajoutera à la réponse le champ « next_page_id », suivi d'un numéro identifiant le reste des informations. Il nous suffit alors de faire une nouvelle requête API avec le paramètre « _page_id={numéro renvoyé} » pour récupérer l'ensemble des données manquantes. Le fonctionnement du programme est illustré par le schéma suivant (*cf. figure 18*).

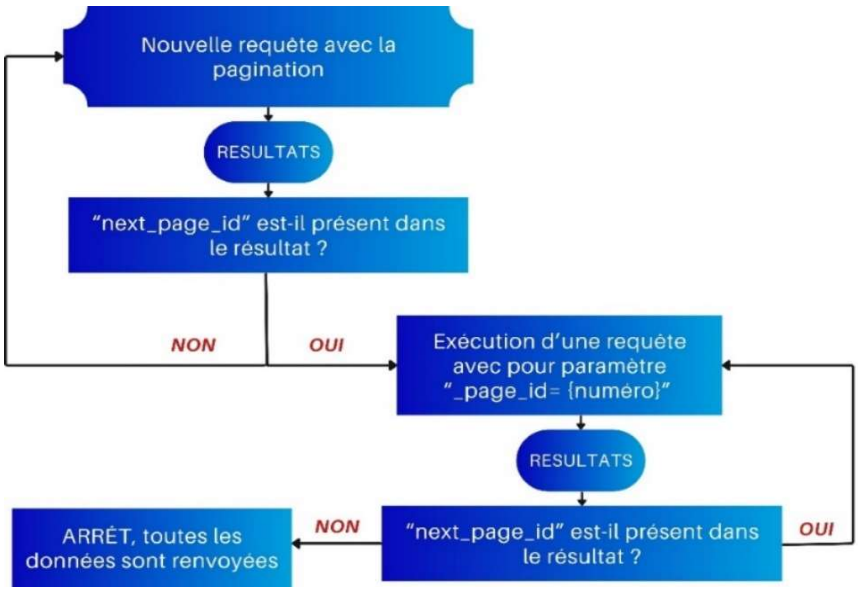

Figure 18 Fonctionnement de la pagination

2.1.7 Conclusion optimisation de la recherche DNS

La recherche récursive des conteneurs et la mise en place de la pagination ont permis de décupler les résultats obtenus, multipliant par 25 le nombre d'enregistrements récupérés, faisant passer le total de 10 521 à 256 677 adresses IP pour le même conteneur. Ma refonte du script est désormais celle adoptée. Ce premier projet m'a non seulement permis de maîtriser de nouveaux outils tels que Tower, Infoblox, et les requêtes API, mais aussi de renforcer et de mettre en pratique mes compétences techniques acquises lors de mon Bachelor, dans le cadre d'un projet professionnel concret.

2.2 Optimisation du suivi des alarmes

2.2.1 Présentation préalable des outils

1. APIC[17] Cisco :

Cisco ACI (Application Centric Infrastructure) est une solution de SDN (Software-Defined Networking) conçue pour les Datacenters. Elle centralise la gestion du réseau à travers l'APIC qui est un contrôleur réseau centralisé qui gère et simplifie l'automatisation et la configuration du réseau. L'APIC joue un rôle crucial en facilitant la mise en œuvre d'une infrastructure agile et adaptative et joue également un rôle essentiel dans la surveillance du réseau, en envoyant des alarmes pour signaler des problèmes ou des changements. Chacun dispose également d'une API, nous permettant d'automatiser certains processus. Nous disposons au total de six APIC, nommés PARIS EQUINIX, AMSTERDAM EQUINIX, AMSTERDAM PRODUCTION, TOULOUSE PRODUCTION, VALBONNE PRODUCTION et VALBONNE VALIDATION dont la disposition est illustrée par le schéma suivant (*cf. figure 19*)

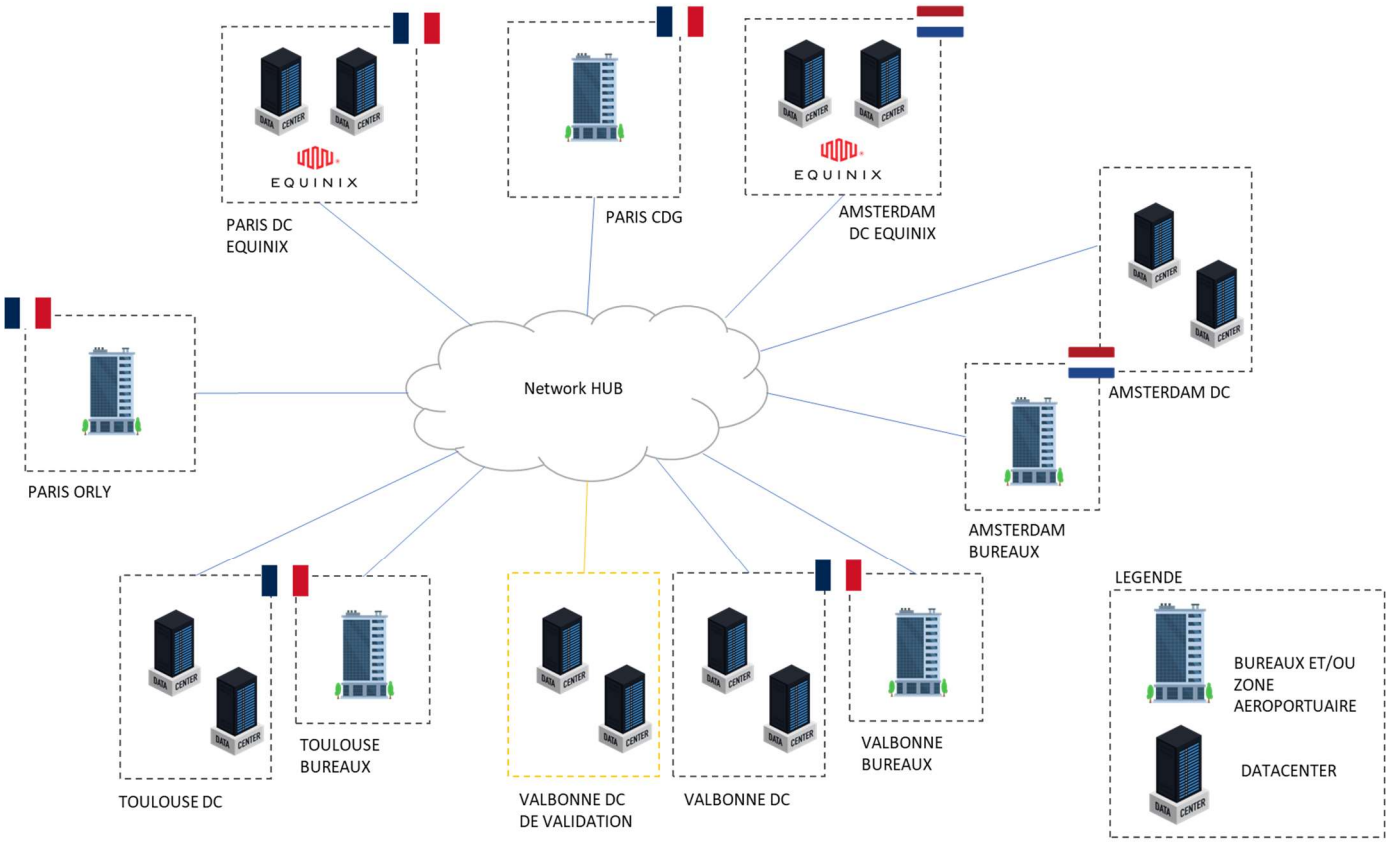

Figure 19 Représentation de la disposition des différents datacenters (DC)

2. NSP :

NSP est un outil interne de centralisation, de visualisation et de gestion des informations réseau, conçu dans un souci de simplification d'accès aux données hébergées par les différentes applications. Grâce à son interface web, il permet une vue rapide et

[17] Application Policy Infrastructure Controller pour Contrôleur de l'infrastructure de la politique d'application.

ergonomique de l'ensemble des outils, et tend à devenir une interface privilégiée pour les différentes équipes, simplifiant leur travail au quotidien. La figure ci-dessous (*cf. figure 20*) représente son fonctionnement et schématise ses interactions avec les différents outils.

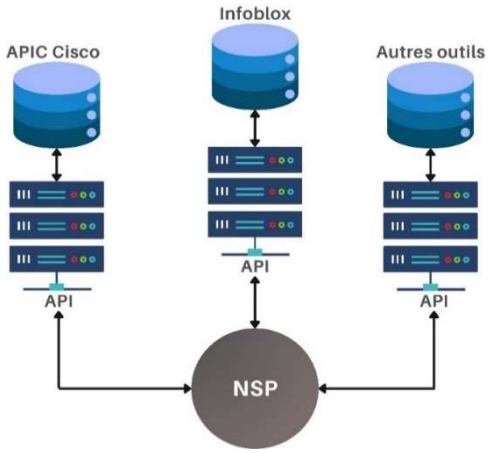

Figure 20 Interaction entre NSP et les différents outils

2.2.2 Présentation des alarmes

Les alarmes sont classées en fonction de leur niveau gravité. On en distingue 4 :

1. *Warning (Avertissement)* : Constitue en une alerte décrivant une situation qui pourrait devenir préoccupante.
2. *Minor (Mineure)* : Concerne des problèmes qui n'impactent pas directement les performances mais qui pourraient nécessiter une attention.
3. *Major (Majeure)* : Signale des problèmes susceptibles d'affecter les performances et nécessitent une intervention rapide.
4. *Critical (Critique)* : Indique des situations urgentes qui affectent les opérations du réseau et nécessitent une attention immédiate.

Il faut également comprendre qu'une alarme est composée d'attributs, tels que sa date de création, sa sévérité ou encore sa cause.

2.2.3 Contexte et objectifs

Les alarmes permettent aux équipes préposées d'identifier rapidement où et quand des problèmes surviennent sur le réseau, permettant une intervention rapide limitant les pannes et autres désagréments. Le suivi des alarmes est donc une nécessité prépondérante dans la gestion des infrastructures réseau. Comme énoncé dans (cf. 1.4 cadre technique général du sujet, pages 6-7), la quantité d'alarmes restituée par les APIC ne permet pas une gestion efficace de celles-ci. Notre objectif est donc de fournir un panel d'outils permettant une gestion globale depuis NSP.

2.2.4 Ajout d'un champ commentaire à chaque alarme

Dans l'état actuel des choses, les APIC et NSP ne permettent pas d'indiquer des commentaires sur les alarmes. Ce manquement entrave la bonne collaboration des équipes qui souhaiteraient pouvoir les annoter de quelques précisions. La communication étant une priorité impérieuse au bon fonctionnement d'un service, un tel ajout pourrait

donc encourager la mise en commun des informations et, plus globalement, faciliter le suivi des alarmes.

Mise en place de la base de données

Pour stocker les données, nous avons décidé de mettre en place une base de données nommée « comments », constituée de quatre tables[18], conformément aux quatre niveaux de gravité. L'intérêt de cette disposition est d'éviter la répétition du niveau de sévérité, puisque le niveau de sévérité est défini une seule fois dans le nom de chaque table. La figure suivante illustre cette structure (*cf. figure 21*).

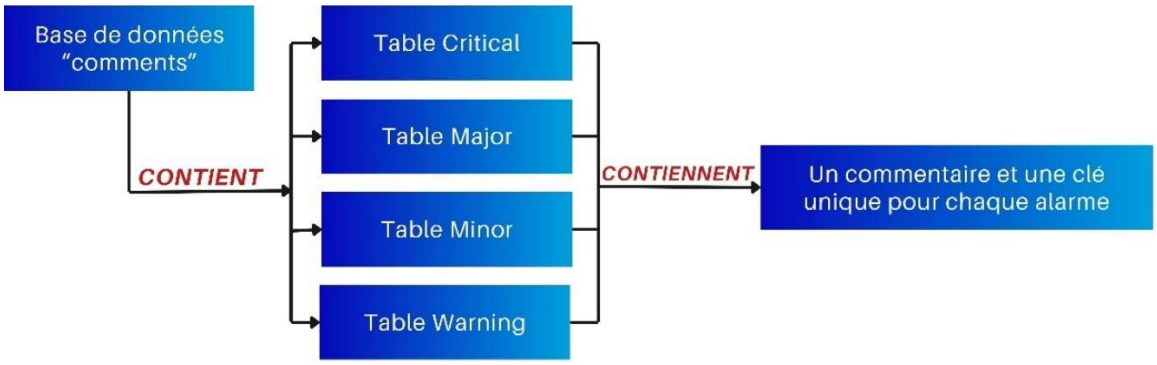

Figure 21 Structure de la base de données « comments »

Identifier les alarmes

Dans la structure de la base de données « comments », chaque commentaire est associé à une clé unique qui identifie l'alarme correspondante. Pour cela, nous devions développer un moyen de générer un identifiant unique à partir de chaque alarme, pour servir de clé unique dans la table.

Les alarmes disposent d'un seul attribut qui garantit leur unicité, nommé « *affected object* » : combinaison d'un ensemble de termes techniques (*cf. figure 22*).

```
uni/infra/funcprof/accportgrp-IPG-SRV-PORT-TLSRBDD7-BOND-T1/rsattEntP/fault-F0989
```

Figure 22 Attribut « affected object » d'une alarme

Bien que cet attribut réponde à notre problématique, la structure résultante est une chaîne de caractères interminable, ce qui n'est pas idéal pour le stockage en base de données ni pour une manipulation ultérieure. C'est pourquoi nous avons décidé de hacher l'attribut en MD5, nous procurant ainsi une structure stable et une chaîne de taille fixe (32 caractères), plus apte au stockage et à l'utilisation.

[18] Une table permet de stocker et d'organiser des données.

Fonctionnement du programme

Nous avons découpé la mise en place de cette nouvelle fonctionnalité en deux étapes. La première étant le développement du programme principal dont le fonctionnement vous sera décrit dans cette partie. La deuxième est l'adaptation du programme à l'infrastructure existante, impliquant de lourdes modifications sur les pages HTML[19] de NSP.

Avant même de commencer son développement, nous avons décidé de diviser le programme en deux fonctions distinctes. L'une subsidiaire, destinée à mettre à jour les commentaires en base de données, et l'autre, principale, chargée de l'ajout et de la suppression des alarmes dans cette même base, ainsi que du renvoi des commentaires. Les alarmes sur lesquelles les fonctions s'exécutent appartiennent au même niveau de sévérité et sont pour l'instant indiquées à la main au sein même du programme. Les figures suivantes représentent individuellement le fonctionnement des deux fonctions du programme (*cf. figure 23 et 24*).

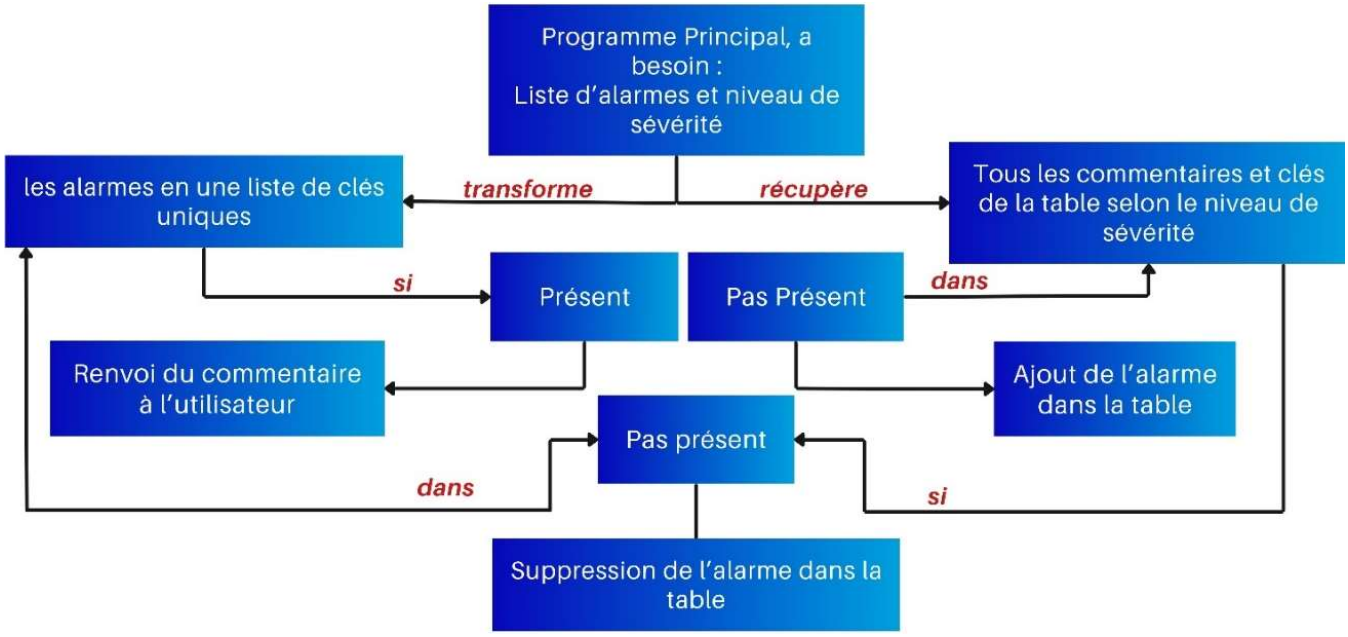

Figure 23 Fonctionnement du programme principal

Figure 24 Fonctionnement du programme secondaire

[19] HTML (HyperText Markup Language) : langage de balisage utilisé pour créer et structurer des pages web.

Mise en place de la solution

NSP disposait déjà de fonctions pour récupérer et trier les alarmes selon leur gravité, produisant quatre pages HTML distinctes pour chaque niveau de sévérité. Intégrer notre script dans cette configuration préexistante a été notre principal défi dans ce projet.

Nous avons commencé par une refonte des pages HTML, ajoutant à chaque page un champ pour les commentaires et un bouton d'enregistrement qui permet aux utilisateurs d'interagir directement avec notre programme. Nous avons également ajouté des gestionnaires d'événements en jQuery[20] pour gérer les interactions avec ces boutons, facilitant ainsi le renvoi des données au programme lorsqu'ils sont sollicités.

Pour la gestion des pages HTML et pour l'envoi des alarmes à celles-ci, nous avons choisi Flask[21] pour sa légèreté et sa modularité. De plus, au cours de mon cursus, j'ai eu l'opportunité de créer des applications web à partir de Flask, ce qui a facilité ma compréhension de la structure globale et m'a permis une intégration moins ardue.

Nous avons également fait face à des défis techniques qui nous ont obligés à revoir notre solution. Par exemple, notre fonction principale pour ajouter et supprimer des alarmes effectuait une requête SQL individuelle pour chacune d'entre elles, ce qui n'est pas problématique lorsque le volume d'alarmes est raisonnable. Cependant, certains niveaux de sévérité comme "Minor" regroupent parfois plus de 900 alarmes, soit autant de requêtes SQL. Cela provoquait des délais significatifs, interdisant toute consultation des pages.

Pour remédier à cela, nous avons rationalisé le processus en regroupant les données avant d'effectuer les requêtes. Cela a réduit le nombre d'opérations à un maximum de trois requêtes : ajout, suppression, et récupération.

L'ensemble du processus est riche en interactions en met en œuvre des notions techniques. C'est pourquoi nous l'avons illustré par des graphiques qui simplifient la compréhension des échanges entre NSP et les fonctions impliquées (*cf. figure 25 et 26*).

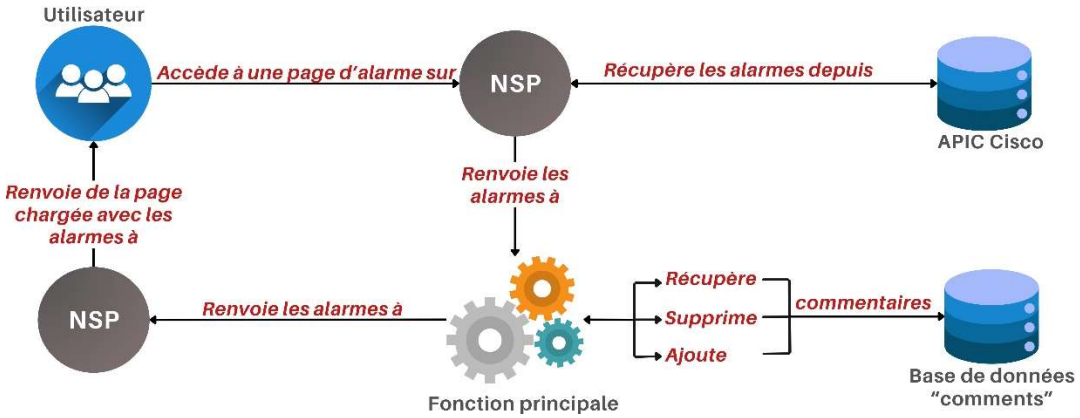

Figure 25 Fonctionnement de la fonction principale

[20] JQuery est une bibliothèque JavaScript légère qui permet de simplifier le développement de scripts côté client en rendant le code plus accessible et moins verbeux.
[21] Flask permet de développer des applications web en Python.

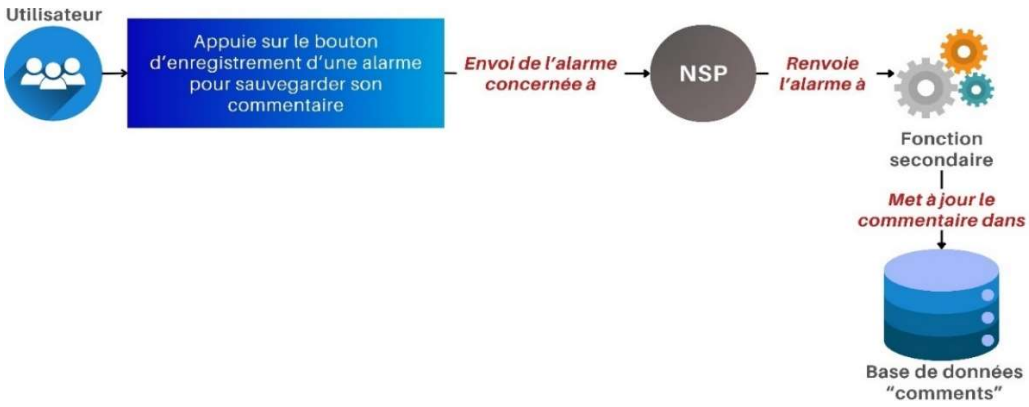

Figure 26 Fonctionnement de la fonction secondaire

2.2.4.1 Résultats et continuité

La figure suivante (*cf. figure 27*) montre ce qui a été ajouté sur NSP (à l'exception du tableau).

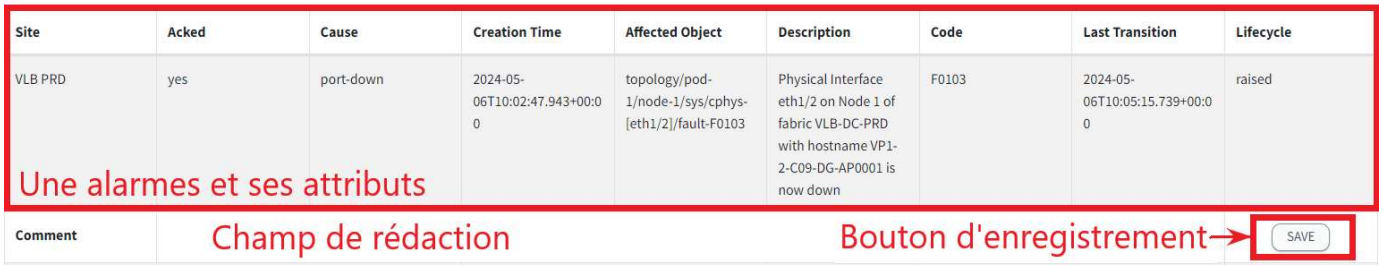

Figure 27 Champ de rédaction et bouton d'enregistrement ajoutés à NSP

Une amélioration possible pour cette fonctionnalité serait la mise en place de la traçabilité des commentaires en intégrant une nouvelle table dans la base de données « comments ». Elle contiendrait les noms des auteurs et les dates de création des commentaires, ce qui faciliterait le suivi des modifications. En outre, l'amélioration du champ de saisie représenterait une évolution intéressante. Actuellement, un commentaire de grande taille pourrait altérer la compréhension du tableau. Il serait donc avantageux d'offrir la possibilité de masquer les commentaires au bon vouloir de l'utilisateur.

2.2.5 Conception d'un graphique

Actuellement, NSP dispose d'un graphique représentant le nombre d'alarmes par APIC (*cf. figure 28*). Celui-ci, jugé peu représentatif, a fait l'objet de nombreuses demandes de modification lors des réunions de services. Le souhait exprimé par les équipes est un graphique triant les alarmes par mois et par APIC, facile à lire, et permettant une représentation claire et explicite. Notre objectif était donc de répondre à cette demande qui, in fine, facilitera le travail des équipes en simplifiant la représentation et le suivi des alarmes.

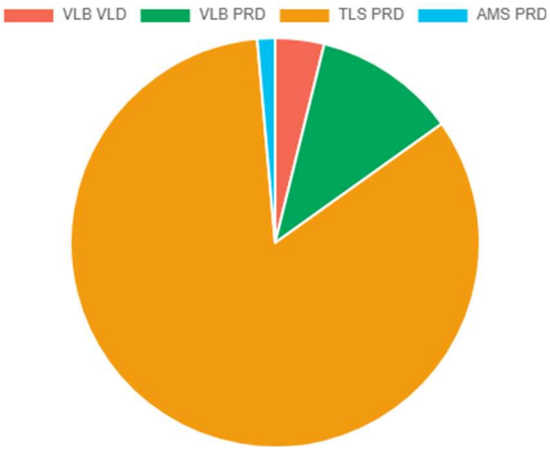

Figure 28 Représentation du nombre alarmes par APIC

Manipulation et création des graphiques

Pour la création et la mise en place des graphiques, nous avons utilisé des exemples proposés par la librairie javascript « *amchart* ». L'utilisation de ces modèles nous a permis de gagner un temps considérable mais également de tester différentes solutions pour répondre au mieux à la demande initiale. Ainsi, nous n'avions qu'à réadapter les paramètres, les données à afficher, et les couleurs. Dans notre conception, nous avons choisi de représenter le nombre d'alarmes en ordonnée[22] et les mois en abscisse[23].

Premiers essais

Nous avons exploré plusieurs types de graphiques en utilisant les scripts fournis par *amchart* pour identifier la meilleure option. Le modèle que nous avons initialement choisi (*cf. figure 29*) semblait prometteur mais présentait, selon nous, un risque de saturation et de perte de lisibilité si de nouveaux APIC étaient ajoutés.

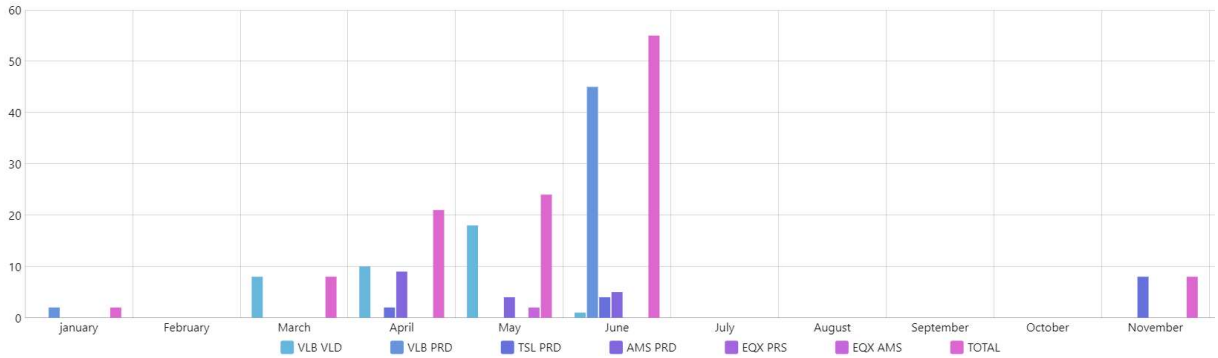

Figure 29 Premier graphique sélectionné

Poursuivant nos recherches, nous avons découvert un format de graphique dit « stacked[24] » qui répondait mieux à nos besoins. Ce nouveau design (*cf. figure 30*) organise les

[22] Il s'agit l'axe y (ce sont les valeurs tout à fait à gauche)
[23] Il s'agit de l'axe x (ce sont les valeurs sous le graphique)
[24] Se traduit par « empilé », ce sont des graphiques en étage.

données par APIC sur différents étages, chaque étage affichant le nombre d'alarmes relevées chaque mois.

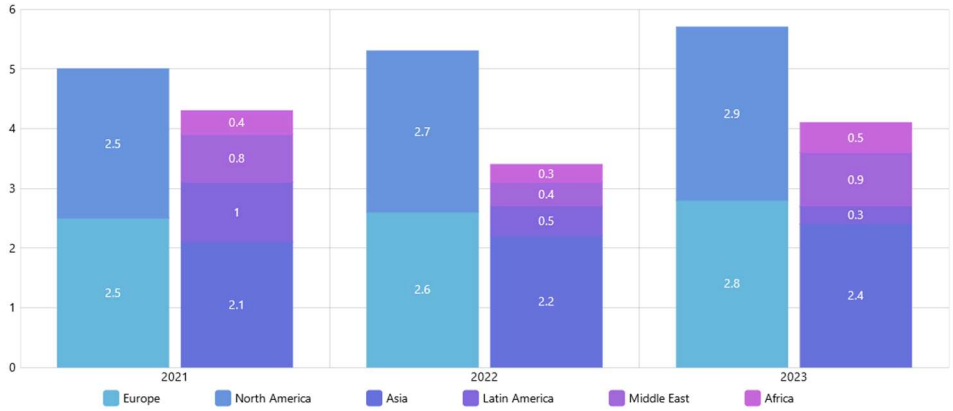

Figure 30 2éme type de graphique sélectionné

Cette disposition nous a convaincus par sa simplicité à transmettre l'information, mais sa mise en place fut particulièrement laborieuse.

Problèmes rencontrés

Bien que le dernier graphique soit idéal par sa disposition, il présente un problème majeur. Lorsqu'une valeur est trop grande par rapport aux autres, les valeurs plus basses se retrouvent écrasées sur l'axe des abscisses, rendant l'ensemble illisible (*cf. figure 31*).

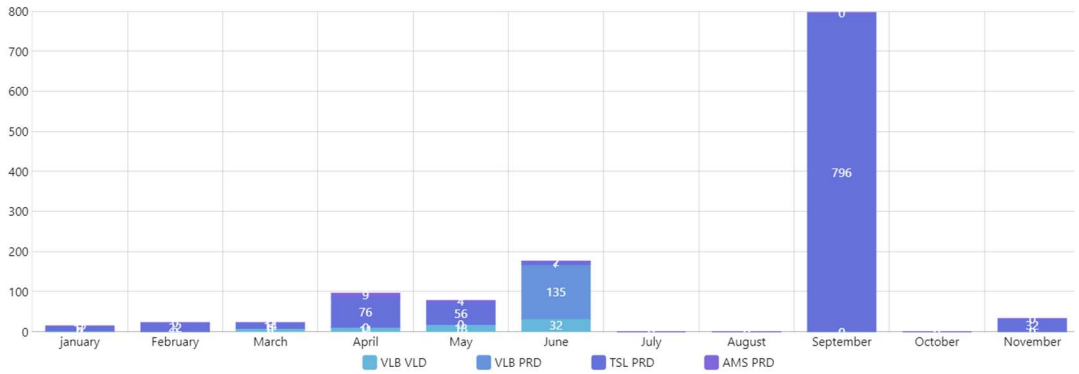

Figure 31 1ére tentative infructueuse avec le graphique en étage

C'est pourquoi nous avons passé un temps significatif pour essayer de trouver une solution convenable. Après un certain temps de recherche, nous avons appris que nous pouvions choisir une échelle logarithmique[25] plutôt que linéaire. Bien que cette approche ait réglé certains problèmes, elle s'est avérée peu pratique car elle déplaçait certains chiffres et rendait l'échelle peu intuitive. Face à cela, nous avons donné aux équipes le choix entre le graphique précédent (*cf. figure 29*) et le nouveau (*cf. figure 32*). Elles ont finalement opté pour ce dernier.

[25] Echelle où chaque unité représente une multiplication par un facteur constant, contrairement à une échelle linéaire où chaque unité représente une addition constante.

Figure 32 Mise en place d'une échelle logarithmique sur le graphique en étage

Optimisation du graphique et résultat final

À la suite de ces ajustements minutieux, nous avons intégré un total avec une taille fixe affiché uniquement sur les colonnes comportant au moins deux étages. Pour alléger visuellement le graphique, nous avons supprimé les zéros et regroupé les APIC par spectres de couleurs similaires. De plus, nous avons ajouté une barre permettant d'agrandir le graphique et fait disparaître les valeurs de l'axe des ordonnées qui ne correspondaient plus aux valeurs affichées. Les valeurs de chaque APIC peuvent également être masquées en cliquant sur leur étiquette. Finalement, cette approche nous a permis d'obtenir le résultat souhaité, comme le montre la figure suivante (*cf. figure 33*).

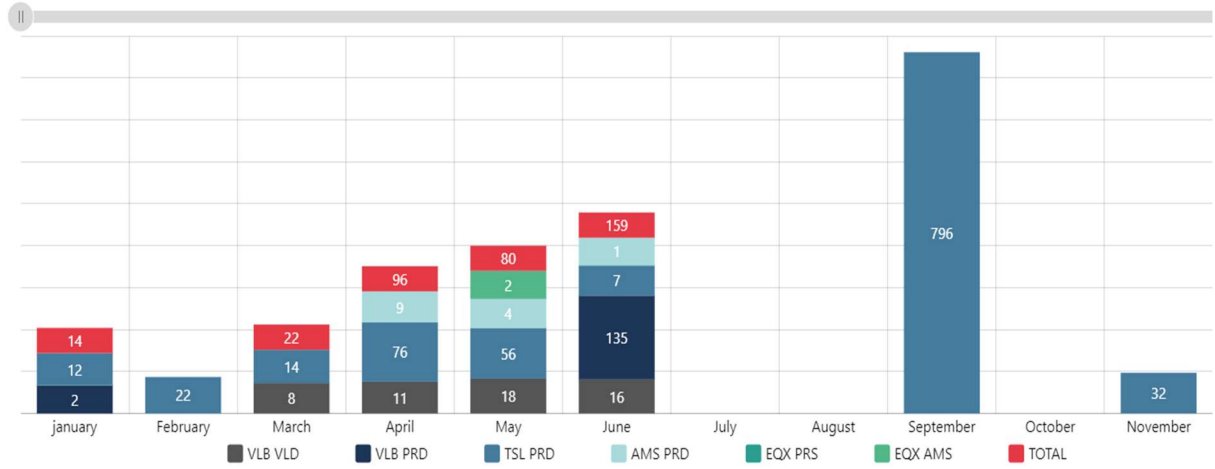

Figure 33 Rendu final du graphique en étage

Continuités et amélioration future

Une idée d'améliorations pourrait être de pouvoir cliquer sur un étage du graphique et d'en avoir un autre qui représenterait quand les alarmes de l'APIC ont été relevées durant le mois. Il s'étendrait sur les jours ou semaines, et permettrait une visualisation précise et dynamique de toutes les alarmes.

2.2.6 Acquittement des alarmes depuis NSP

L'acquittement permet de marquer une alarme comme étant traitée ou en cours de traitement. Cette fonctionnalité, primordiale à leur suivi, constitue une étape obligatoire pour chaque alarme. Actuellement, l'acquittement nécessite une connexion obligatoire à l'APIC qui l'a relevé, un processus chronophage compte tenu de la quantité d'alarmes et de la multiplicité des APIC. Cette procédure pourrait être grandement simplifiée par l'intégration de cette fonctionnalité directement depuis NSP, complémentant ainsi le champ « commentaire » qui permet aux équipes de préciser l'état et l'avancement de la résolution.

Fonctionnement et mise en place

Tout comme les parties précédentes, l'acquittement sera effectué via une requête API. À la seule différence que l'on envoie des données au lieu d'en récupérer. La requête a besoin de deux paramètres : le premier est l'attribut « affected object », qui garantit l'unicité de l'alarme comme abordé dans la partie *cf. 2.2.4.2* identifier les alarmes.
Le deuxième est l'attribut « *ack* », qui est associé à l'état de l'acquittement, celui-ci prend deux valeurs : « yes » lorsque l'alarme est acquittée et « no » quand elle ne l'est pas. L'envoi se fait par le biais d'une fonction prédéfinie dans NSP. L'on peut observer dans les données qu'elle envoie l'attribut « ack », le reste est quant à lui passé en paramètre au sein de l'URL (*cf. tableau 3*).

Données envoyées	URL avec le paramètre « affected object » [26]
`{"faultInst": {"attributes": {"ack": "yes"} }}`	`URL_APIC/api/node/mo/uni/<affected_object>`

Tableau 3 Données et URL envoyés lors d'une requête API d'acquittement

Nous avons adapté la fonction d'envoi existante dans NSP pour gérer les requêtes d'acquittement et de non-acquittement. En outre, une « checkbox »[27] a été ajoutée à côté de chaque alarme sur l'interface NSP, permettant aux utilisateurs d'acquitter ou de révoquer les alarmes facilement (*cf. figure 34*).

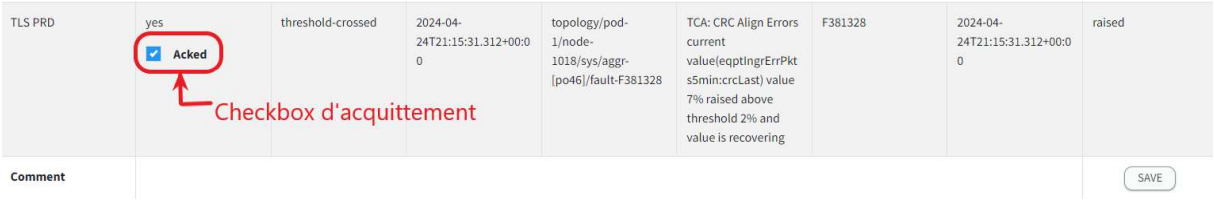

Figure 34 Checkbox d'acquittement ajoutée à NSP

L'étape la plus ardue a été la mise en place de l'échange entre l'utilisateur, NSP et le script, ce que nous avons illustré par le graphique suivant *(cf. figure 35*).

[26] Les autres paramètres « node », « mo », « uni » permettent de préciser le chemin d'accès à l'alarme.
[27] Case à cocher utilisée pour sélectionner ou désélectionner une option.

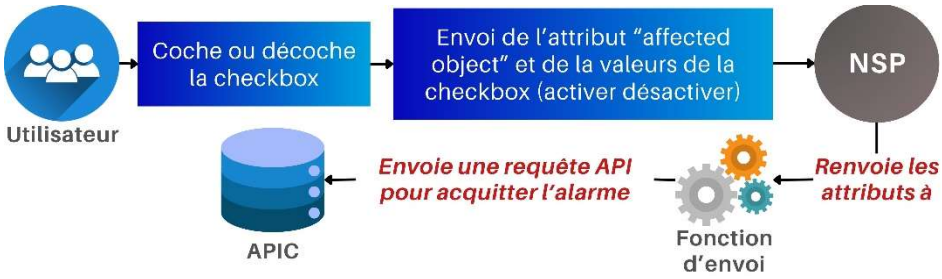

Figure 35 Interaction entre le programme, NSP et l'utilisateur

2.2.7 Conclusion optimisation et suivie des alarmes

Ce deuxième projet m'a permis de jouer un rôle direct dans la gestion des alarmes et dans l'optimisation du travail des équipes. J'ai pu conforter mes compétences et mes connaissances dans les requêtes API vues lors du premier projet, comprendre et manipuler de nouveaux outils comme les APIC Cisco et NSP, et mettre en pratique des connaissances vues lors de mon Bachelor comme la création d'applications web et la gestion de bases de données. Quant à la pérennité du projet, d'autres fonctionnalités pourraient être ajoutées, comme la possibilité d'afficher les alarmes selon des critères de filtrage précis.

3 Conclusion générale

En conclusion, mon stage au sein du groupe Air France KLM a été une expérience très enrichissante. J'ai eu l'opportunité d'explorer différents aspects du monde professionnel, tout en développant mes compétences personnelles.
Au cours de mon stage, j'ai travaillé sur plusieurs projets concrets, notamment la refonte d'un script de recherche DNS, et l'optimisation du suivi des alarmes, tous deux s'inscrivant dans le contexte de l'automatisation des sanity checks. Ces activités, progressivement plus fortes en responsabilités, m'ont permis d'appliquer mes connaissances théoriques et pratiques et m'ont appris à respecter des cahiers des charges, à organiser efficacement mon travail.

Grâce à mon stage, j'ai pu observer de près le fonctionnement d'une entreprise aussi grande que complexe et comprendre comment les différents services et activités s'intègrent dans des objectifs communs.

En analysant les résultats obtenus dans mes projets, j'ai pu constater l'impact direct de mes contributions sur les équipes, notamment sur la gestion des alarmes où j'ai pu directement contribuer à l'optimisation de leur travail. Ces résultats ont renforcé ma conviction quant à l'importance de la communication et de la coopération non seulement au sein d'une équipe mais également entre les différents services.

En matière d'apprentissage personnel, ce stage m'a permis de développer des compétences clés telles que la gestion du temps, due aux différents projets sur lesquels j'ai travaillé, la communication en milieu professionnel et le travail en équipe grâce aux réunions régulières auxquelles j'ai pu participer activement, notamment en y partageant mon travail et en recueillant des axes d'amélioration. J'ai également appris à maîtriser de nouveaux outils comme les requêtes API, les APIC Cisco, Tower Ansible et Infoblox, me donnant ainsi un bagage technique et fonctionnel valorisant vis-à-vis de mes futures expériences professionnelles.

En conclusion, ce stage a été formateur et m'a permis d'acquérir des compétences techniques et professionnelles ; tout au long de celui-ci, j'ai eu la nette impression de d'être plus qu'un stagiaire mais de faire partie intégrante d'une équipe qui a toujours su apporter une valorisation à mon travail.

4 Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à mon maître de stage, Monsieur Sercer Romain, pour son accueil, ses conseils et le partage de son expertise, mais également pour la confiance qu'il m'a accordée tout au long de ce stage.

Je remercie également Monsieur Blais Antoine, pour sa patience et pour son formidable esprit d'équipe.

Je tiens à remercier Monsieur Dolique Stéphane, responsable de l'équipe ITDS TZ, pour avoir accepté de m'intégrer au sein de son équipe et pour l'attention qu'il a portée au bon déroulement de mon stage.

Je souhaite également remercier toute l'équipe ITDS TZ pour leur bienveillance et leur accueil. Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage.

Bibliographie

1.4 Cadre général du sujet note 7. Récupéré sur https://geekflare.com/fr/netops-explained/

1.4 Cadre général du sujet, note 6. Récupéré sur https://www.atlassian.com/fr/itsm/incident-management/itscm

2.1.1 Description préalable des outils, Redhat ansible tower. Récupéré sur https://people.redhat.com/mkoch/training/1806-vienna/presentations/05%20-%20Ansible%20and%20Ansible%20Tower%20Introduction.pdf

2.1.5 Présentation préalable des requêtes API. Récupéré sur https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces

2.2.1 Présentation préalable des outils, ACI. Récupéré sur https://www.cisco.com/c/fr_fr/solutions/collateral/data-center-virtualization/application-centric-infrastructure/solution-overview-c22-741487.html

2.2.1 Présentation préalable des outils, APIC cisco. Récupéré sur https://www.cisco.com/c/fr_ca/products/cloud-systems-management/application-policy-infrastructure-controller-apic/index.html

2.2.4 Ajout d'un champ commentaire à chaque alarme, note 19. Récupéré sur https://fr.wikipedia.org/wiki/Hypertext_Markup_Language

2.2.4 Ajout d'un champ commentaire à chaque alarme, note 20. Récupéré sur https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203587-jquery-definition/

2.2.4 Ajout d'un champ commentaire à chaque alarme, note 21. Récupéré sur https://datascientest.com/avantages-et-fonctionnement-de-flask