Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Développement de scripts d'automatisation & Immersion dans le service Support Client

Lucas RIPOCHE

JAGUAR NETWORK

Responsable entreprise : Yoann Normand

Responsable académique : Arnaud Février

2019

Table des matières

Introduction :

Le DUT Réseaux et Télécommunications a pour but d'apprendre les fondamentaux théoriques de l'administration réseaux et systèmes. Mais aussi de mettre en pratique ces connaissances, aussi bien au cours de TPs que lors d'un stage de fin d'études de 10 semaines permettant de valider toutes ces connaissances théoriques et pratiques.

J'ai effectué mon stage de fin d'études en tant que technicien au sein de la Cellule de Support Client de la société Jaguar Network, sous la tutelle de Yoann Normand, Administrateur Système au sein du CSC. Mon projet était de développer un script pouvant émettre des recommandations sur la configuration d'Apache et Nginx en fonction de certains paramètres.

Mon stage s'est découpé en deux activités. Le matin était dédié au traitement de tickets afin de résoudre les différentes problématiques des clients de la société. L'après-midi était quant à elle dédiée au développement des différents scripts d'automatisation de tâches.

Cette immersion au sein de l'équipe du support m'a permis de découvrir les différents aspects de l'activité d'opérateur télécom et d'hébergeur. Cela m'a aussi permis d'apprendre beaucoup sur le déploiement d'une infrastructure, aussi bien d'un point de vue réseaux que système. De plus, cela m'a permis de développer mes compétences en relationnel avec des clients.

Ainsi nous allons d'abord détailler l'historique, le panel d'activité ainsi que l'organisation de Jaguar Network. Ensuite, une partie sera dédiée à expliquer l'activité de support que j'ai effectuée. Par la suite, nous allons détailler l'utilité, les choix techniques ainsi que les méthodes de développement des scripts réalisés lors de mon stage.

Présentation de l'entreprise :

1) Description

Jaguar Network est une société fondée en 2001 par Kevin Polizzi à Marseille. Historiquement, son activité principale est l'hébergement de site web. En 2006, Jaguar Network devient un Opérateur Télécom orienté vers le marché professionnel. En 2013, Jaguar Network inaugure son premier datacenter à Marseille. Depuis, la société a développé un réseau fibré de plus de 7000 Km. Jaguar Network a aussi englobé plusieurs entreprises, telles que Alionis, VoIP 720° et Online Telecom. Depuis Janvier 2019, Jaguar Network fait partie du groupe Illiad, ce qui ouvre une nouvelle phase de croissance.

2) Panel d'activités

Jaguar Network est notamment implanté dans 4 domaines.

Le premier domaine, le plus important, est les solutions Cloud. Jaguar Network a développé une offre Cloud hybride, permettant de s'intégrer dans n'importe quelle infrastructure IT existante.

Le second domaine est les Télécoms. Comme dit précédemment, Jaguar Network possède un réseau fibré à l'échelle européenne de 7000km. Ainsi, la société peut proposer du transit inter-datacenter, des solutions d'interconnexion, des réseaux privés ainsi que de la Voix sur IP.

Le troisième domaine comprend les solutions IoT et Big Data. Jaguar Network met à disposition de client des composants logiciels. Ainsi la société propose des moteurs de recommandations utilisant des technologies du Big Data et du Machine Learning, en interne et en externe.

Le quatrième domaine est les services managés. La société propose à ses clients des offres d'infogérance. Que ce soit au niveau matériel, en fournissant des serveurs et du stockage. Mais aussi au niveau logiciel, ne proposant la gestion d'applications, de virtualisations, d'APIs ou encore de cloud. La société propose aussi des conseils pour la mise en conformité au GDPR.

La clientèle de Jaguar est constituée exclusivement d'entreprises, des PME ainsi que des grands groupes. Tous les clients ont une architecture sur mesure et accès au support 24/7, toute l'année.

3) Organisation interne

Ainsi on peut distinguer plusieurs services au sein de l'entreprise.

Le Département Administratif, Financier et Juridique a pour de gérer tous les aspects financiers au sein de la société. Ce service doit notamment contrôler la cohérence et la faisabilité du modèle économique de la société en fonction de sa croissance. Il doit aussi assurer le contrôle de revenus du groupe et gérer les différents aspects juridiques. Enfin ce service doit gérer l'allocation des ressources financières au sein du groupe et mettre en place la stratégie financière et fiscale de développement du groupe à l'international.

Le département Commercial doit développer le chiffre d'affaire de la société, que ce soit en démarchant de nouveaux clients, en accompagnant les clients dans les modifications de leurs infrastructures ou encore d'informer la clientèle des nouvelles offres proposées.

Le département des Opérations représente le cœur d'ingénierie de la société, ce service a pour buts de garantir la continuité d'activité du développement technique des différents projets de Jaguar Network. Mais aussi d'architecturer, d'intégrer et de déployer les infrastructures Jaguar Network ainsi que les plateformes et solutions clientes.

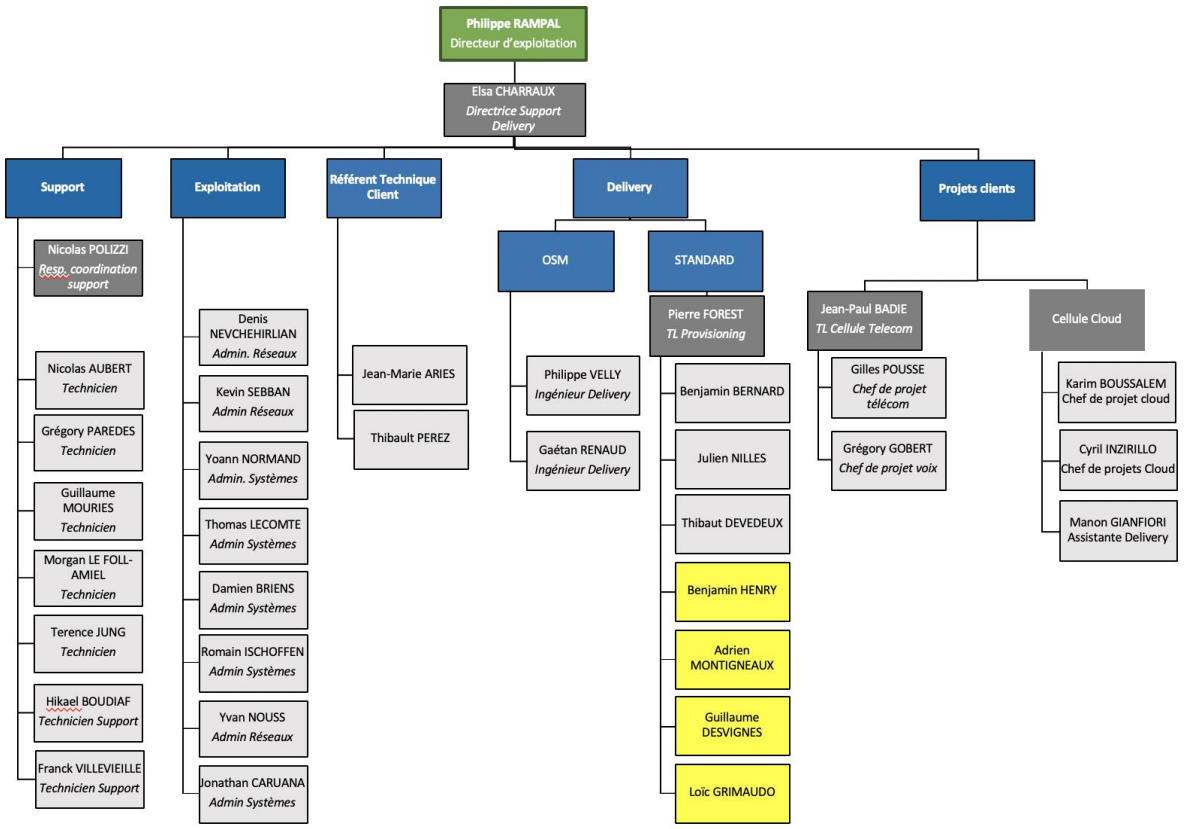
Le département du Système d'Information doit développer l'entièreté des outils permettant de collecter, stocker, traiter et distribuer les différentes informations afin d'améliorer les performances et maximiser l'efficacité des collaborateurs.

Le département Support et Delivery a pour but d'assurer le déploiement des offres vendues aux clients. Mais aussi en assurer leur exploitation et le support des infrastructures en place. C'est au sein de ce service que j'ai effectué mon stage.

Le département des Ressources Humaines doit assurer le recrutement des collaborateurs en fonction des demandes exprimés par les managers des différents services. Ce service doit aussi gérer la carrière des collaborateurs pour accroître leurs talents techniques et managériaux. Enfin ce service doit assurer la gestion des relations humaines et sociales au sein de l'entreprise.

Le département du Marketing et de la Communication doit assurer l'évolution du portefeuille d'offres JN et promouvoir tant l'image générale du groupe Jaguar Network que ses différentes offres. Ce service gère aussi bien la communication externe qu'interne.

Comme dit auparavant, j'ai effectué mon stage au sein de la Cellule Support Client du service Support et Delivery. L'organigramme du service support clientèle est le suivant :



4) Culture d'entreprise

La culture d'entreprise au sein de Jaguar Network est basée sur le partage et l'excellence. Les services sont organisés dans des plateaux permettant facilement l'échange entre les collaborateurs. La montée en compétence est un point important, ayant permis à bon nombre d'employés d'évoluer au sein de la société.

Immersion dans l'activité de Support Client :

Mon stage avait un double objectif, d'abord la réalisation de projets d'automatisations que nous allons détailler plus tard. Ensuite je devais aussi assurer une fonction de support client. Cela impliquait de la gestion de tickets, la prise d'appels et parfois une escalade technique.

Je vais donc commencer par expliquer les différents types de tickets, présenter l'outil de gestion des tickets et enfin détailler la classification des tickets.

1) Principe

Tout client souscrivant à une offre Jaguar Network a accès à un extranet sur lequel ils peuvent visualiser les différents services souscrits, leurs états actuels, l'ouverture de tickets ainsi que le suivi de ceux-ci. Il existe deux types de tickets :

- Le premier type de ticket est celui ouvert par le client. Il caractérise souvent un problème ou une prise d'information. Chaque ticket doit être émis par un compte extranet certifié. Dans le cas où le client appelle directement, dans la plupart des cas un ticket est créé pour la traçabilité.
- Le deuxième type de ticket est celui ouvert par l'équipe Jaguar Network à destination du client. Ce sont la plupart du temps des tickets dus à des évènements en supervision. Par exemple, une surcharge anormale des ressources ou une application qui n'est plus à jour. Ils peuvent être soit créé automatiquement par le système de supervision soit directement par un employé.

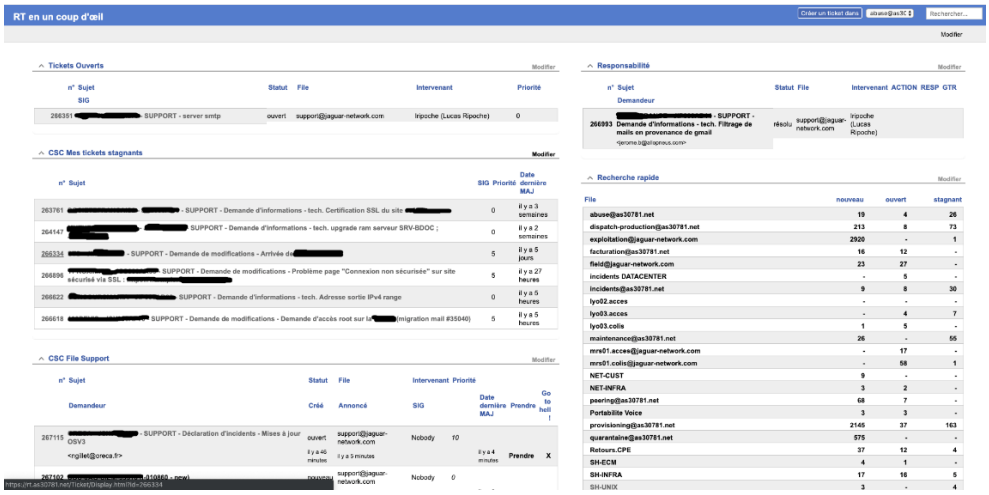
A noter que ce système de ticket est très important du point de vue de la traçabilité et de la protection légale. Ces échanges de mails prouvent ce qui a été faits et permettent parfois la résolution postérieure de problèmes. Ainsi il est aussi important de commenter les tickets en listant toutes les actions effectuées.

2) Fonctionnement

Dans le concret, cet outil est matérialisé par deux sites web.
Le premier étant l'extranet client dont voici l'interface :

Extranet Client - Figure 1

Le deuxième est l'outil RT permettant le suivi des tickets par le support.



Outil RT - Figure 2

L'outil RT :

L'outil RT permet de suivre trois files :

- Les nouveaux tickets
- Les tickets ouverts, c'est-à-dire ceux que l'on traite et qui sont en attente d'une réponse
- Les tickets stagnants, étant en attente d'une réponse du client

On peut aussi distinguer plusieurs catégories de tickets se répartissant en deux divisions : Réseaux et Systèmes.

Nous allons donc les lister :

- Tickets orientés Réseaux :
  - Des lignes fibres ou ADSL coupées
  - Changer, enlever ou ajouter des règles de Firewalling.
- Tickets orientés Système :
  - Mise en place de certifications SSL
  - Redirection de sites
  - Gestion du stockage des machines
  - Création de nouveaux utilisateurs

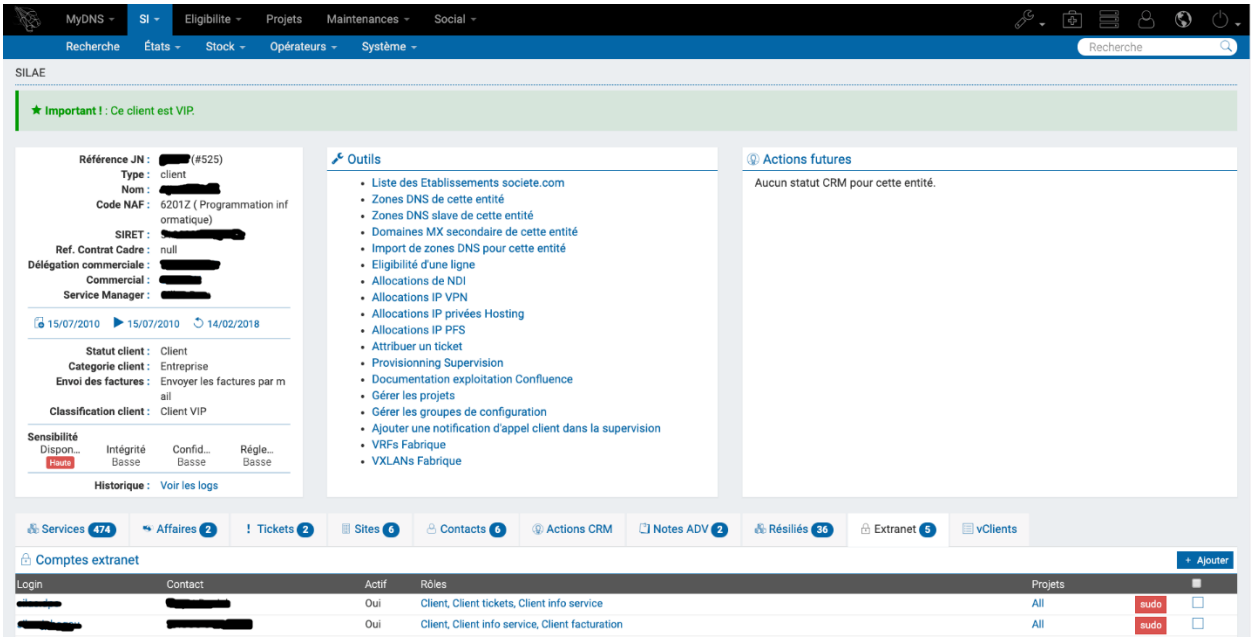
Le reste des tickets concerne des problèmes réseaux ou système plus spécifiques. Pour les traiter il faut donc prendre connaissance de l'architecture en place. On peut soit se renseigner directement auprès du responsable de la plateforme. Soit consulter la documentation sur le Wikipédia interne. Cela est aussi valable pour les tickets systèmes.

3) Les outils

Il est aussi intéressant de lister les autres outils disponibles aux membres du Support.

Le Système d'Information :

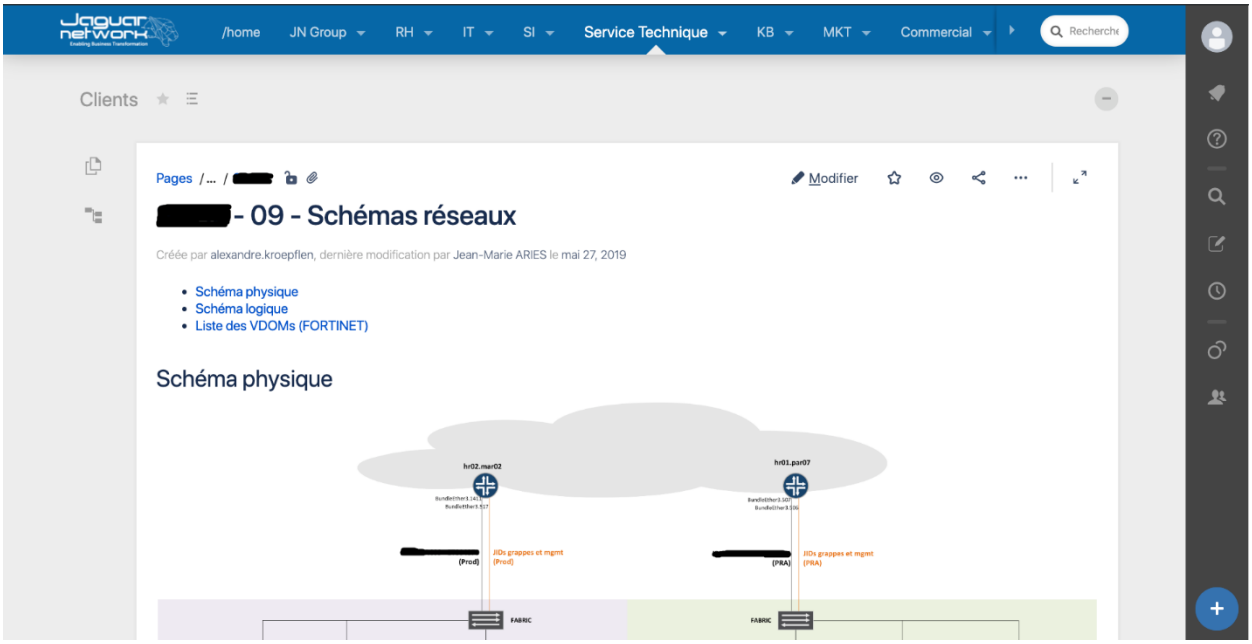
Un outil très important est le Système d'Information, il permet d'accéder à toutes les informations relatives aux clients. C'est ici que l'on peut voir tous les services proposés aux clients, les informations de connexions à chaque équipement, des graphiques d'utilisation des connexions et des machines virtuelles et enfin toutes les informations de contact de l'entité cliente.



Système d'Information - Figure 3

Confluence :

Enfin, le dernier outil très important au sein du support est Confluence. C'est le Wikipédia interne. Il contient la documentation de toutes les offres, tous les services, toutes les infrastructures JN, le fonctionnement d'un grand nombre de technologies utilisés au sein de la société, de la communication interne et externe mais surtout de toutes les infrastructures clientes. Ainsi cet outil est utile lorsque l'on doit se renseigner sur un client, son architecture et les technologies utilisées.



Confluence - Figure 4

En conclusion, l'activité de support m'a permis de renforcer mes compétences en réseaux et en système mais aussi de découvrir l'infrastructure de Jaguar Network. Elle m'a aussi permis de découvrir les différentes procédures en place au sein de la société et de mieux comprendre le rôle de chaque service.

Développement de scripts d'automatisation :

Ainsi comme expliqué auparavant, Jaguar Network connaît une forte croissance et a besoin d'automatiser un grand nombre de tâche répétitive afin de pouvoir assurer plus facilement le support de toutes les infrastructures clientes. C'est pourquoi les scripts que j'ai réalisés ont tous pour objectif d'aider à dresser un diagnostic sur certains points d'une machine cliente. De plus, l'immersion au sein de l'activité support m'a permis de mieux comprendre le fonctionnement global du service et ses actions afin de mieux cibler ce que mes scripts devaient traiter.

I) Développement de scripts Bash

Comme évoqué auparavant, nous allons à présent détailler le développement des deux premiers scripts réalisés durant mon stage.

1) Analyseur de partitions :

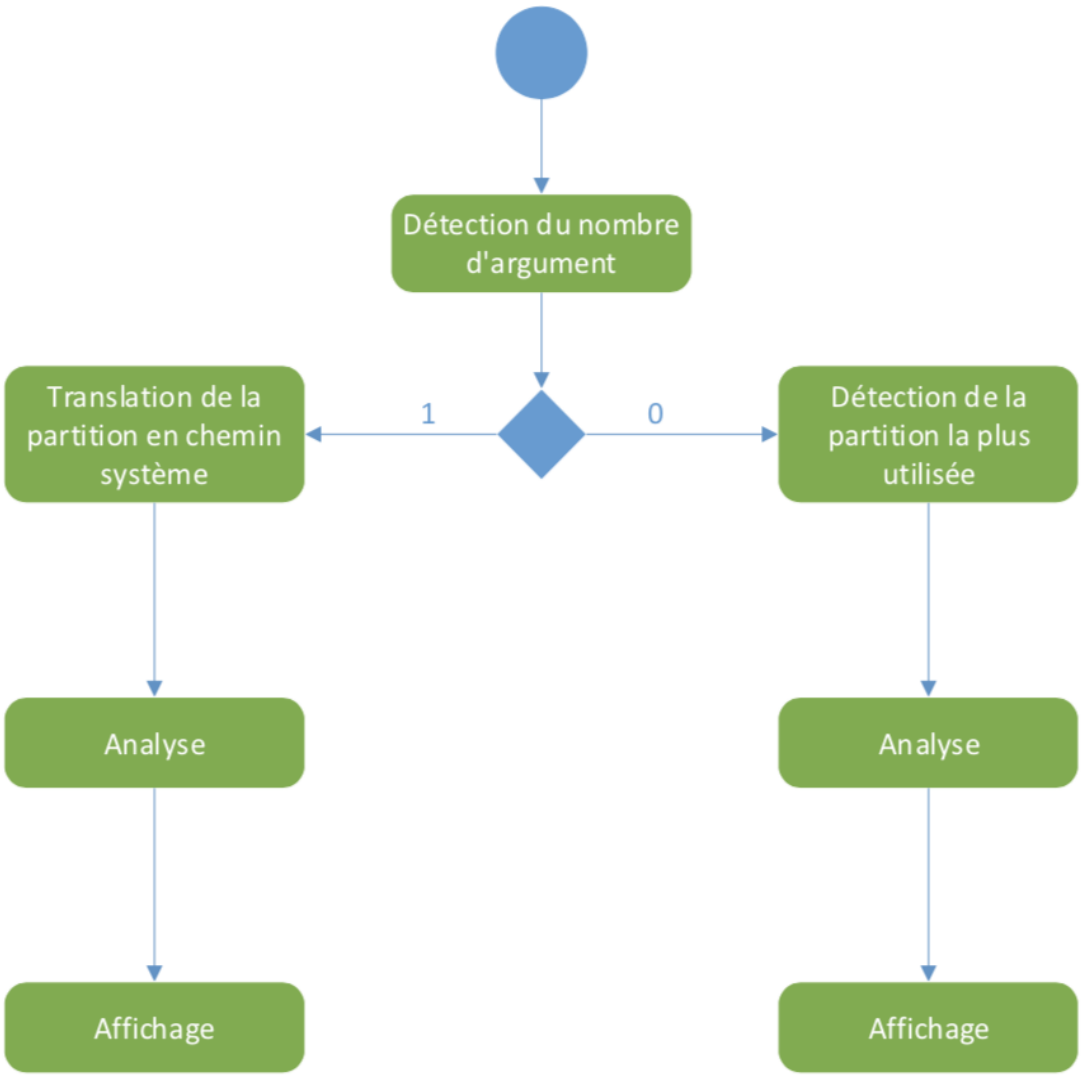
Le premier script que j'ai eu à réaliser est un script devant détecter les dix fichiers les plus volumineux d'une partition. En effet, actuellement les membres du support doivent exécuter eux-mêmes un jeu de commandes pour identifier puis analyser les différentes partitions. Il faut souvent analyser plusieurs machines d'un même client, cela est donc une opération répétitive et chronophage. Ainsi, le script doit avoir deux modes d'opération, le premier étant le mode automatique. Dans ce mode le script détecte automatiquement la partition la plus occupée et l'analyse. Le second mode est le mode manuel. Dans ce mode, une partition est donnée en argument et le script doit l'analyser.

Pour le développement du script nous avons choisi Bash car le script doit être léger à l'exécution, s'adapter à un grand parc linux et ne doit pas avoir besoin d'un interpréteur pour fonctionner. De plus, Bash est un langage conçu pour le Scripting sur Linux, son utilisation est donc simple.

Une fois le script exécuté, sa première action va être de vérifier le nombre d'arguments de la commande d'exécution.
Si le script est exécuté sans argument alors le script va appeler la fonction la fonction *check_part()*. Cette fonction a pour but de lancer une suite de commandes afin de déterminer la partition la plus remplie. Pour cela il se base sur une commande permettant de connaître le pourcentage d'utilisation d'un espace de stockage. La sélection se fait donc en fonction du pourcentage d'utilisation et non pas du disque ayant la volumétrie la plus importante. Une fois que nous avons récupéré la chaîne de caractères représentant le nom de la partition, nous devons trouver le point de montage* associé. Cette association est trouvée par un jeu de commandes, nous stockons le résultat dans la variable "info". Ensuite le script appelle la fonction *analyse()*.

Cette fonction va lister les fichiers de plus de 10 MégaOctets et va les classer par ordre de taille décroissant. Toutefois, nous avons instauré un garde-fou au cas où la partition contienne plus d'un million de fichiers, cela se matérialise par une condition *if*.

De plus, la fonction doit détecter puis afficher si la partition contient des fichiers dont l'état est "deleted"*. Une fois cela fait, la fonction exécute un jeu de commandes permettant de lister les 10 fichiers les plus volumineux de la partition, puis procéde à l'affichage, ce qui termine le script.



2) Analyseur de fichiers de journalisation :

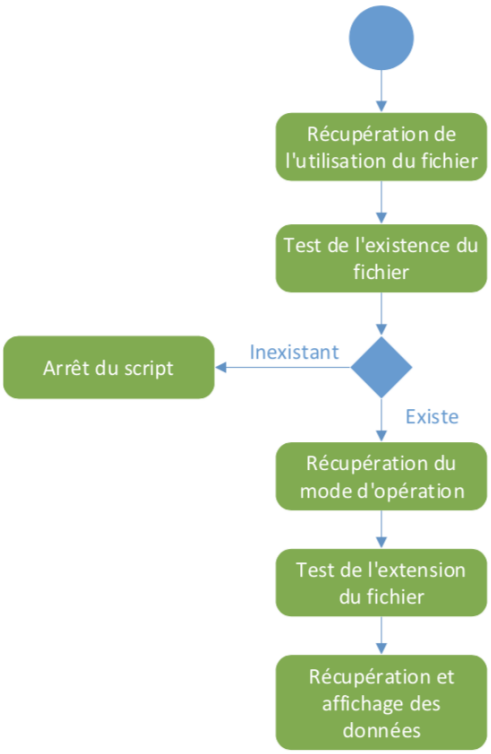
Par la suite m'a été confié le développement d'un autre script. Ce script a pour but de récupérer certaines statistiques à partir d'un fichier de journalisation.
Le script doit pouvoir récupérer trois statistiques différentes :

- La première est un classement des dix adresses IP ayant fait le plus de requête au serveur.
- La deuxième permet d'afficher le nombre de requête au fil des minutes d'une heure et d'une date donnée.
- La troisième permet d'afficher le nombre de requête au fil des heures d'une date donnée.

Une contrainte technique supplémentaire est que le script doit pouvoir analyser les fichiers compressés au format .gz.

Pour ce projet nous avons utilisé Bash pour les mêmes raisons citées précédemment.

Cette fois-ci le script doit se lancer sans arguments. Ainsi il va commencer par demander à l'utilisateur le chemin du fichier. Ensuite le script va lancer la fonction *file_exists()*. Cette fonction va arrêter le script si le chemin spécifié par l'utilisateur n'existe pas. Une fois ce test effectué, la fonction va afficher les différentes options possibles puis va enregistrer la réponse de l'utilisateur. Ceci fait, le script va exécuter la condition concernée. Ainsi, dans tous les cas, le script va détecter l'extension du fichier et va donc exécuter des commandes adaptées pour la lecture du fichier compressé ou non. Dans le cas du traitement en fonction de la date ou de l'heure, le script va récupérer ces variables auprès de l'utilisateur. Ensuite le script va lancer un jeu de commandes pour récupérer les informations voulues, puis les afficher.



II) Projet de recommandations Nginx et Apache

Par la suite m'a été confié le développement d'un script plus ambitieux, l'optimisation de la configuration d'un serveur web. Ce projet est motivé par un besoin fort. En effet Jaguar Network héberge de nombreux sites web de petite taille. Tous ces sites web réunis consomment d'importantes ressources. Il est donc intéressant d'optimiser le fonctionnement des serveurs web animant ces sites web.

1) Besoin

Le but de ce projet est de développer un script permettant, lors de la phase de création ou de production d'un serveur Apache ou NginX, de détecter la configuration physique et la configuration logicielle du serveur pour faire des recommandations sur la configuration du serveur web.

2) Cahier des charges et choix techniques

Nous allons donc détailler le cahier des charges de ce projet.

La première contrainte est les distributions linux à supporter. L'environnement Jaguar Network étant majoritairement composé de machines opérant sur Debian 8 et 9 et sur Ubuntu 18.04. Il faut donc supporter ces distributions mais aussi prendre en compte le futur déploiement de Debian 10.

En ce qui concerne le choix du langage, il fallait trouver un langage stable, sans interpréteur, fonctionnant sur un grand nombre de distributions et pérenne dans le temps. Bash fut immédiatement exclu du fait des changements majeurs de fonctionnement entre ses différentes versions, ce qui nuit à la pérennité du script. Il a fallu ensuite faire un choix entre Pearl et Python. Mon choix s'est porté sur Python car il est plus moderne que Pearl et sa communauté a produit énormément de documentation sur tous les sujets. Il est toutefois intéressant de noter que Pearl est réputé plus robuste que Python, de plus son intégration aux environnements Linux est plus aisée. Les deux options étaient donc viables.

Par la suite une nouvelle problématique a fait son apparition. En effet, Python a récemment connu une mise à jour majeure de sa syntaxe, changeant ainsi le fonctionnement de certaines fonctions. Il fallait donc faire un choix entre Python 2 et Python 3. Ayant fait le choix de Python du fait de sa modernité, je me suis naturellement orienté vers Python 3. Ce choix est d'autant plus pertinent que la majorité des machines du parc Jaguar Network vont bientôt supporter uniquement Python 3, et plus spécifiquement Python 3.5.

Un autre choix technique impactant fut le choix du module utilisé pour exécuter les commandes UNIX. En effet, une grande partie du projet repose sur l'exécution de ces commandes, il était donc important de choisir un module pérenne et sécurisé.
La première option était d'utiliser le module *os*, c'est le module historique. Sa syntaxe est simple et complète. Il permet d'effectuer un grand nombre d'opération avec beaucoup de flexibilité. Toutefois depuis python 3 son utilisation est dépréciée.
La deuxième option est le module *subprocess*, sa syntaxe est plus complexe mais elle est globalement plus optimisée. En effet le module *os* s'appuie sur *subprocess*. Pour des raisons de performances et de stabilité il vaut donc mieux utiliser directement *subprocess*.

3) Phase de recherche

Ainsi une fois le cahier des charges et les choix techniques définis il a fallu faire des recherches sur les différents paramètres impactant les performances d'un serveur web. Il fallait répondre à plusieurs problématiques telles que :

- Quels sont les paramètres propres aux serveurs web impactant les performances ?
- Quels sont les paramètres du noyau UNIX impactant les performances ?
- Quelles sont les valeurs de la configuration matérielle à prendre en compte et comment définir des paliers de performance ?
- Doit-on adapter les paramètres de la même façon au vu du volume devant être traité par le serveur web ?

Ainsi mes recherches se sont concentrées sur les paramètres de Nginx afin de réaliser une branche fonctionnelle.

J'ai débuté mes recherches via les documentations officielles Nginx relatives à l'amélioration des performances. J'ai donc produit une documentation des différents paramètres UNIX et Nginx permettant un gain de performances.

Nous allons donc débuter par les paramètres UNIX importants lorsque l'on essaye d'améliorer les performances.

Gestion des files d'attentes du noyau UNIX

Le premier champ d'amélioration possible est la gestion des files d'attente du système et notamment ces deux paramètres :

net.core .somaxconn : Nombre maximal de connexions pouvant être mise en attente par le noyau UNIX. L'augmentation de ce paramètre est importante lorsque l'on reçoit un grand nombre de connexions.
Une bonne condition pour déterminer si ce paramètre doit être augmenté est de vérifier la présence de messages d'erreur dans les fichiers de journalisation du noyau UNIX.

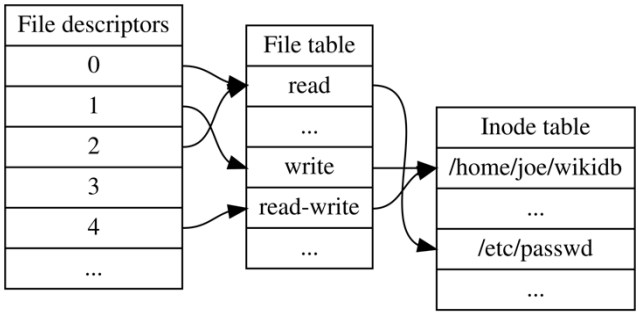
net.core.netdev_max_backlog : Taux auquel les paquets sont mis dans le cache par le réseau avant le traitement CPU. Une augmentation peut augmenter les performances sur une machine disposant d'une bande passante importante.

Ici encore, la présence de messages d'erreur dans les fichiers de journalisation du noyau UNIX est

Le deuxième champ important est la gestion des File descriptors*, et notamment grâce à deux paramètres :

sys.fs.file-max : La limite du nombre de fichiers pouvant être ouvert en simultané par le système.

nofile – Permet de modifier le nombre de file descriptor pouvant être ouvert par une application. Dans le cas présent nous avons fixé la limite pour Nginx à 200 000 fichiers.



Le dernier champ important de modification est la gestion des ports ouverts par le système d'exploitation. Un paramètre en particulier nous intéresse :

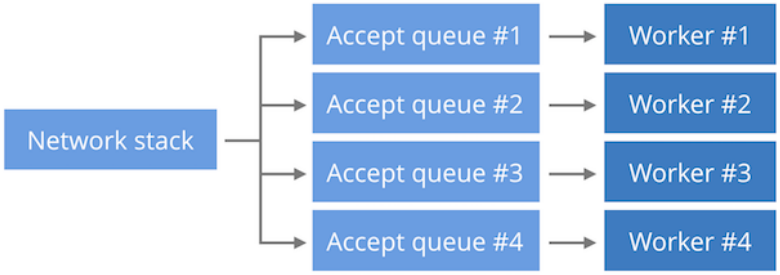
net.ipv4.ip_local_port_range : Plage de ports pouvant être utilisés par les différentes applications présente sur le système. Une pénurie de ports peut entrainer une mise en attente de certaines applications. Ce paramètre est donc très impactant. Une valeur correcte est 1024 à 6500.

Paramètres propres à NGINX :

Nous allons à présent détailler les différents champs de modification important lors de la configuration de NGINX.

Worker Mangement :

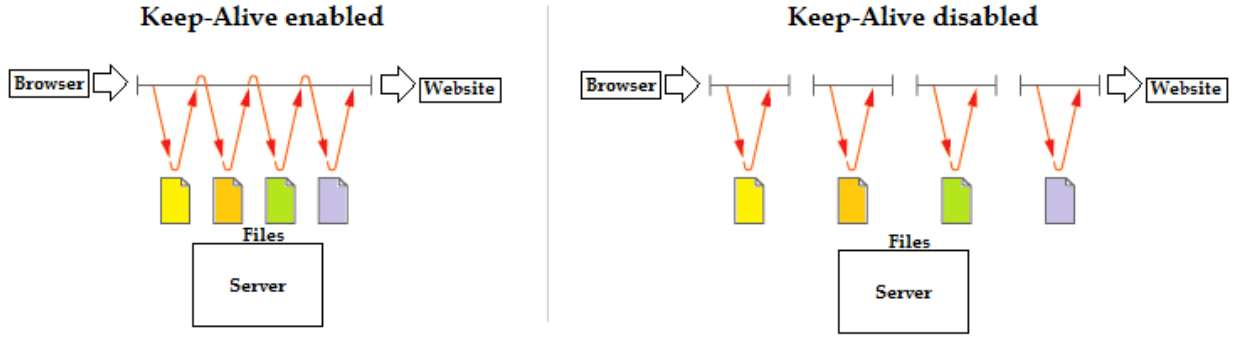
Worker_processes : Ce paramètre permet de gérer le nombre de processus Nginx qui vont être mis en fonction sur le système. Il doit être égal au nombre de cœurs du CPU, il est recommandé de mettre ce paramètre sur auto, il va ainsi détecter automatiquement le nombre de cœurs et s'adapter. S'il y a un grand nombre d'accès I/O* à faire, il peut être intéressant d'augmenter ce nombre afin d'optimiser l'utilisation du disque.



Worker_connections : Par défaut, un worker peut supporter jusqu'à 512 connexions mais cela dépend grandement de la configuration physique de la machine. Les différents paliers peuvent être trouvé après un grand nombre de tests.

Keepalive Connections :

Nous allons maintenant aborder les KeepAlive Connections, ce type de connexion est apparue avec la version 1.0 du protocole HTTP. Si cette option n'est pas activée, pour chaque requête et réponse http, le système va créer une nouvelle connexion TCP pour la transmission. KeepAlive permet de maintenir une connexion pour transmettre plusieurs requêtes et réponses http avec un même client. Cette fonctionnalité permet de faire d'importantes économies de puissance de calcul ainsi que de bande passante.



keepalive_requests : Nombre de requêtes que peut faire un client sur une KeepAlive Connection (par défaut : 100). Un nombre bien plus élevé permet de mieux résister aux tests de charges qui envoient un grand nombre de requêtes en provenance d'un même client.

Keepalive_timeout : Temps où une KeepAlive Connection inutilisée est maintenue par le système d'exploitation.

Log Buffering :

Le log buffering permet de stocker les logs reçus dans la mémoire vive puis l'écrire par morceaux conséquents sur la mémoire physique. Cela permet d'économiser des accès I/O* et de maximiser l'utilisation des ressources système.

Ainsi nous pouvons fixer des valeurs déterminant la taille du cache et la période maximale entre deux mises en cache. Cela s'applique à **access_log** et à **error_log**.

A noter qu'il est aussi possible de désactiver la journalisation même si cela est déconseillé. De plus il est recommandé de journaliser uniquement les erreurs critiques.

Compression :

La compression permet de réduire considérablement l'utilisation de la bande passante mais peut augmenter drastiquement l'utilisation des ressources. Toutefois, aujourd'hui la compression est devenue une norme afin d'éviter la congestion des réseaux.

Nous avons donc décider d'appliquer un jeu d'instructions classique afin de compresser toutes nos ressources de manière optimale.

Il est intéressant de noter que lors des tests que j'ai réalisés la compression n'a pas réellement impacté les performances.

TLS SSL :

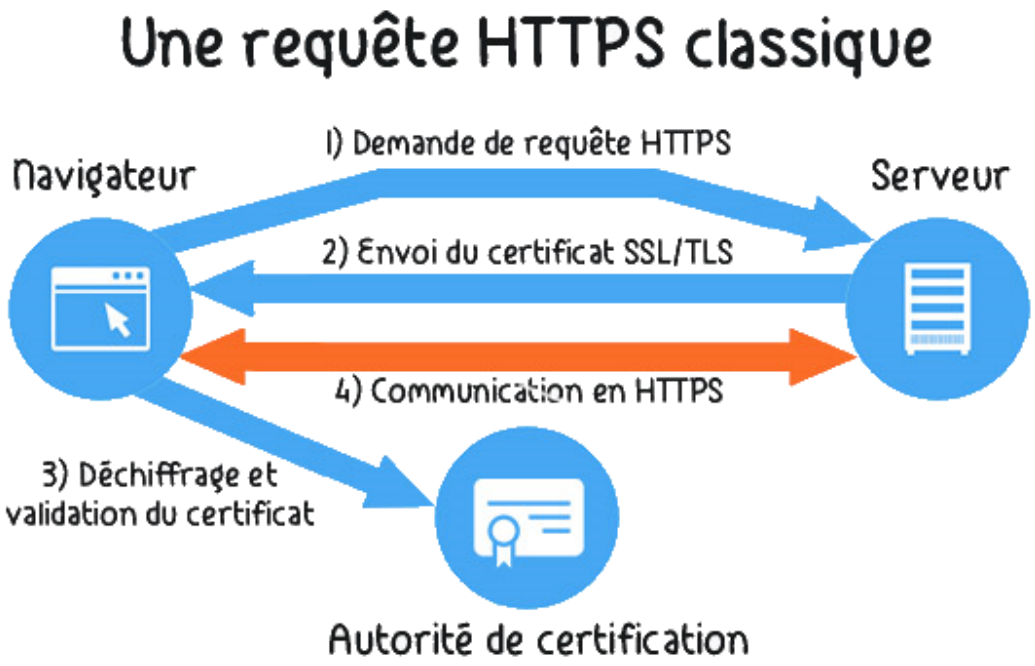
Une modification de configuration dégradant fortement les performances mais qui est vitale pour la sécurité des sites web hébergés est l'activation du protocole TLS SSL.

Ainsi l'activation de TLS (Transport Layer Security) et de son prédécesseur SSL (Secure Sockets Layer) permet d'activer une couche de sécurité supplémentaire. Son fonctionnement suit un mode client-serveur. Ces protocoles vont donc :

- Authentifier le serveur hébergeant le site web via un certificat fourni par une organisation officielle, ce qui assure sa véridicité.
- Le chiffrage des données transmises, ce qui garantit leur confidentialité.
- L'intégrité des données échangées

Afin d'éviter de modifier le fonctionnement du protocole HTTP, un nouveau protocole fut créé, le protocole HTTPS.

Nous détaillerons plus tard la mise en place du protocole TLS SSL.



Fonctionnement du protocole HTTPS – Figure 5

En conséquence nous allons énumérer les différentes optimisations possibles relatives à ces protocoles.

Session Caching : Ce paramètre permet de mettre le certificat SSL en cache et de le mutualiser entre tous les worker processes. Ainsi chaque nouvelle connexion sera plus rapide, cela améliorera aussi les performances sur les autres requêtes faites par le même utilisateur. Cela évite de nombreux accès I/O* devant lire ce certificat. Toutefois il faut définir une taille de cache servant à contenir les différents certificats des différents worker processes ainsi qu'une période de temps servant au renouvellement du cache.
La documentation officielle recommande un cache de 20 MegaOctets et un timeout supérieur à 10 minutes.

Désactiver SSL : En réalité la connexion sécurisée est gérée par TLS, SSL est un protocole souffrant de plusieurs failles de sécurité. Toutefois, la désactivation de SSL créera des incompatibilités avec certaines navigateurs trop anciens ne supportant pas TLS, par exemple Internet Explorer 6. Le public impacté étant réduit, on peut donc sans grand risque désactiver SSL.

Cipher suites : Ce paramètre permet de mettre en place une hiérarchie dans les technologies de chiffrement à utiliser. Cela va être déterminé en fonction de ce que le navigateur de la connexion cliente est capable de gérer. Cela permet de toujours chiffrer les données les données avec la technologie la plus optimisée gérée par le navigateur client.

DHE handshake : Permet d'activer le DHE, c'est un protocole permettant de négocier de manière sécurisé le protocole de chiffrement à utiliser. Cette modification diminue légèrement les performances mais permet une sécurité accrue.

OCSP : Permet d'enregistrer la validation par l'OCSP (Online Certificate Status Protocol) de notre certificat. Si cette option n'est pas activée, à chaque nouvelle requête HTTPS, le client va interroger l'OCSP pour valider que le certificat du site n'est pas falsifié. L'activation de ce paramètre permet de diminuer grandement le temps de traitement de chaque requête.

Strict Transport Security : Sur les navigateurs récents et si cette option est activée, le navigateur n'essaiera plus jamais de résoudre votre site en HTTP et passera directement par HTTPS. Ainsi cela évite le traitement de la redirection au serveur web.

HTTP 2 : L'activation de HTTP2 permet une grande amélioration des performances. En effet la nouvelle version de HTTP2 permet, entre autres, de multiplexer des flux de données. Ainsi lorsqu'un client doit charger une page contenant plusieurs dizaines de ressources différentes, au lieu de créer une connexion à la fois avec un délai entre chaque connexion, HTTP2 va pouvoir multiplexer jusqu'à 6 flux simultanés, ce qui va réduire le temps de chargement des pages et maximiser l'utilisation des ressources du serveur.

4) Réalisation de tests sur les performances

Outils de test :

Une fois ces recherches effectuées, il a fallu mesurer l'effet sur les performances des différents paramètres. Le but était de recréer un cas concret, nous avons donc dédié une machine virtuelle à ces essais. Cette machine est constituée de deux cœurs virtuels, de 4 GigaOctets de mémoire vive, de 20 GigaOctets de mémoire SSD et d'une connectivité à internet de 1 GigaBits par seconde. Cela correspond à la majorité des machines hébergeant des sites web. La machine a pour système d'exploitation Debian 9.1 et utilise Nginx version 1.10.3.

Pour que les résultats des tests soient le plus proche de la réalité que possible, le site hébergé sera un site Wordpress. En effet un grand nombre des sites hébergés par Jaguar Network sont des sites utilisant des CMS tels que Wordpress, Shopify ou Magento. De plus ces CMS nécessitent l'utilisation de bases de données, ce qui complexifie les requêtes. Ceci réduit l'écart type entre tous les résultats des tests, l'installation de Wordpress a réduit de 30% les écarts de temps de traitement entre les différents tests. Les résultats sont donc beaucoup plus précis.

Nous avons aussi mis en place une version HTTPS du site ainsi qu'une redirection du site HTTP vers HTTPS.

Stress Test :

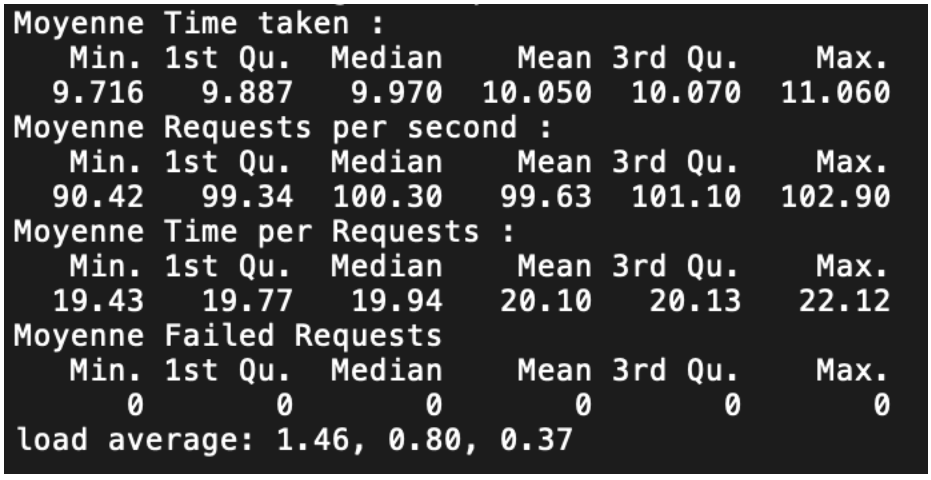
Pour effectuer les tests plusieurs choix étaient possibles, ApacheBenchmark, Siege et ApacheJMeter. ApacheJMeter est un programme très complet permettant de simuler véritablement une utilisation cliente, avec la possibilité de créer des scénarios d'utilisation permettant d'obtenir des résultats plus consistants et précis. Toutefois sa complexité étant trop importante, nous avons décidé de ne pas l'utiliser.
ApacheBenchmark et Siege permettant d'obtenir des résultats similaires, nous avons décidé d'utiliser ApacheBenchmark du fait de sa syntaxe simple.

Le stress test est composé de 10 phases, chaque phase consiste à effectuer 1000 requêtes via 2 connexions concurrentes.

Monitoring :

Pour monitorer ce test nous avons développé un script permettant, depuis une machine distante, de lancer les 10 phases du test puis de recueillir une liste de valeur dans un fichier. Nous récupérons le temps de traitement moyen de toutes les requêtes, la moyenne du temps de traitement par requête, la moyenne du nombre de requêtes traitées par seconde, la moyenne des requêtes n'ayant pas aboutis et la moyenne d'utilisation des ressources système.

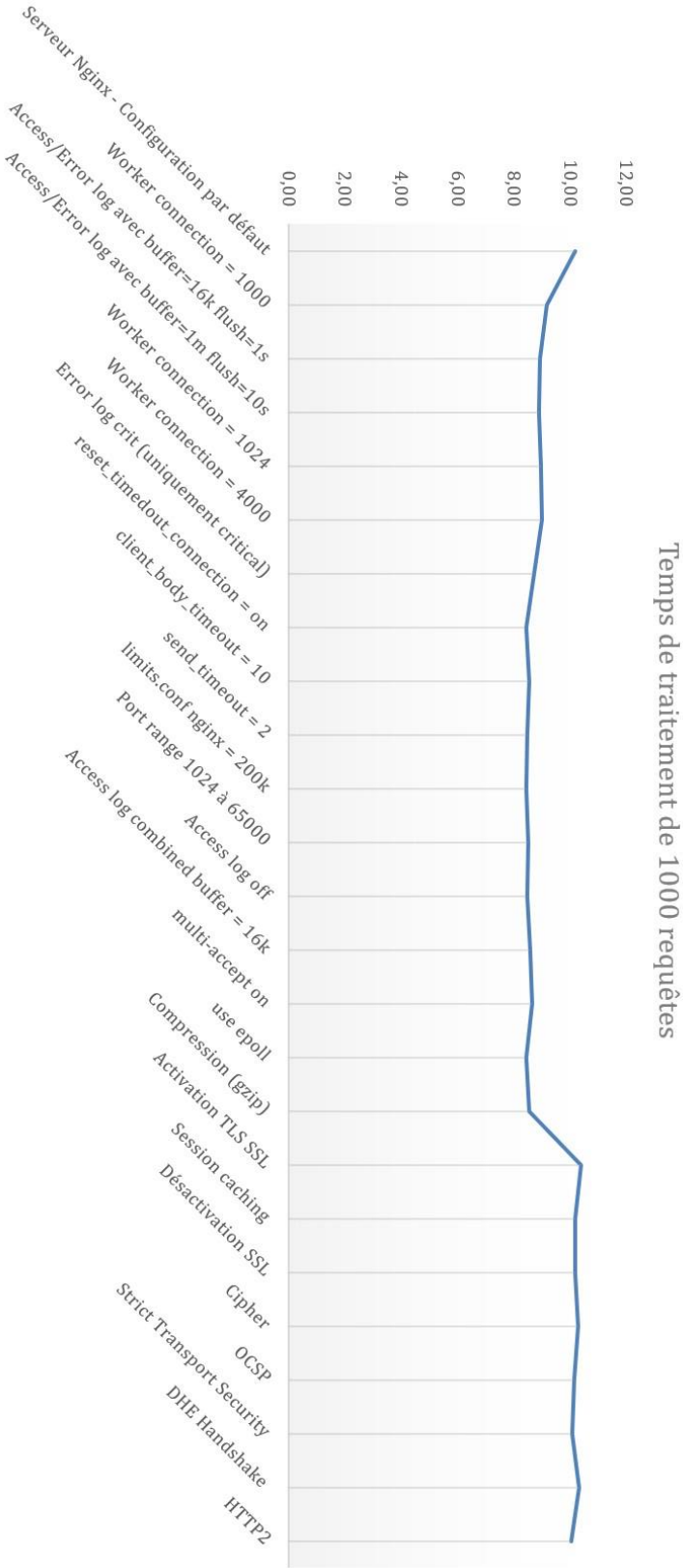
Nous avons realisé 25 tests correspondant aux 25 modifications que nous avons effectuées sur la configuration du serveur Nginx.



Exemple de résultat d'une phase de test – Figure 6

Résultats :

Nous allons à présent détailler les résultats de ces tests.


Temps de traitement de 1000 requêtes
0,00
2,00
4,00
6,00
8,00
10,00
12,00
Serveur Nginx - Configuration par défaut
Worker connection = 1000
Access/Error log avec buffer=16k flush=1s
Access/Error log avec buffer=1m flush=10s
Worker connection = 1024
Worker connection = 4000
Error log crit (uniquement critical)
reset_timedout_connection = on
client_body_timeout = 10
send_timeout = 2
limits.conf nginx = 200k
Port range 1024 à 65000
Access log off
Access log combined buffer = 16k
multi-accept on
use epoll
Compression (gzip)
Activation TLS SSL
Session caching
Désactivation SSL
Cipher
OCSP
Strict Transport Security
DHE Handshake
HTTP2

Nos modifications ont donc permis une augmentation de 19% des performances avant l'implantation de HTTPS. Une fois HTTPS implanté, le gain de performance redescend à 1%. Ainsi nos modifications permettent globalement d'absorber la mise en place de HTTPS.

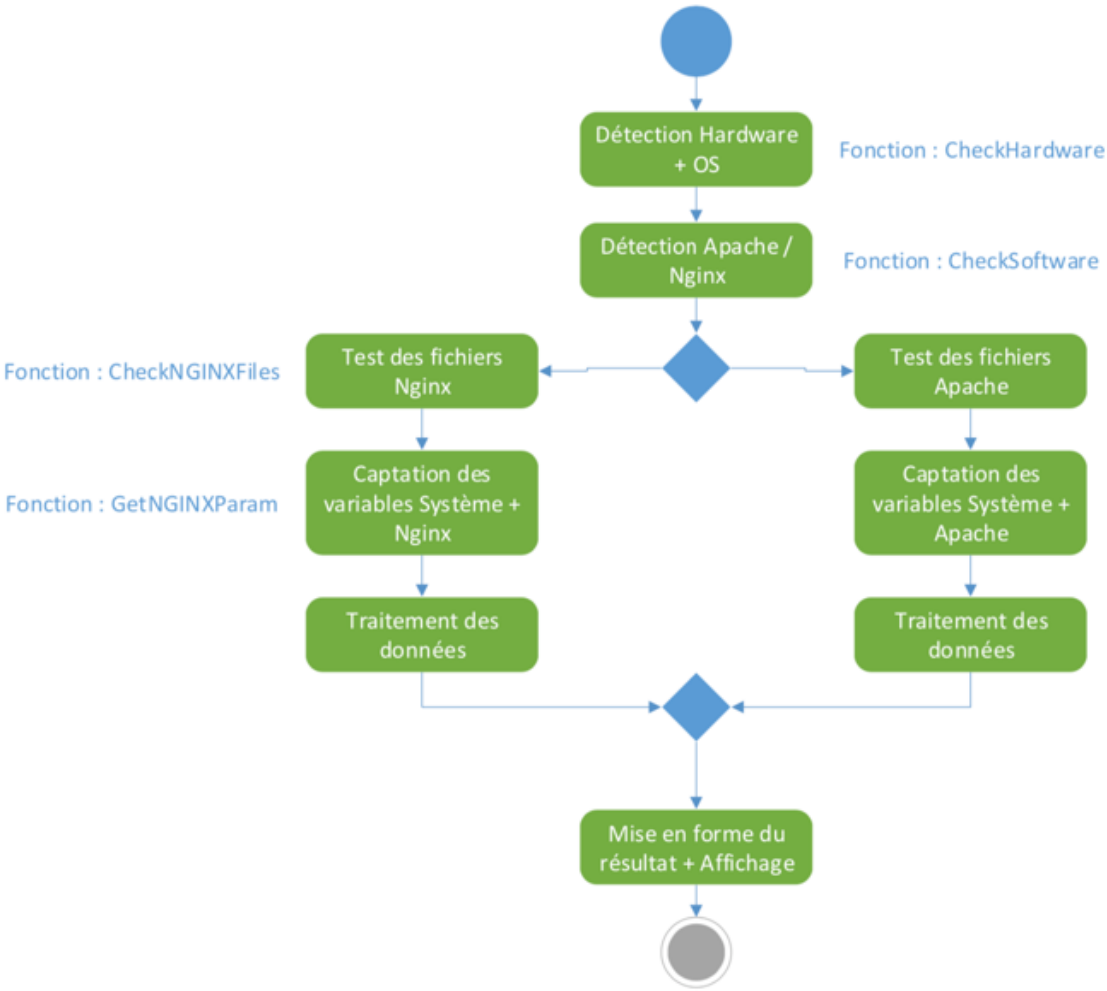
Toutefois il est intéressant de noter que ces modifications appliquées à une machine plus puissante devant traiter plus de requêtes permettraient un gain de performances plus élevé. En effet certains paramètres ne prennent leur sens que lorsque les volumes à traiter sont importants. Par exemple, l'implémentation de HTTP2 s'est matérialisée par une faible augmentation des performances de 2,69%. En effet le site testé est composé de peu de ressources à distribuer, le multiplexage est donc peu utile. Le site TuneTheWeb a réalisé un test consistant à mesurer le temps de chargement d'un site web contenant 36 images statiques en utilisant HTTP, HTTPS puis HTTP2. Selon ce test, HTTP2 est 84% plus rapide que HTTP et 85% plus rapide que HTTPS.

On peut donc imaginer qu'avec l'augmentation des volumes, les modifications appliquées permettraient toujours d'obtenir des performances satisfaisantes.

5) Développement du script

Une fois ceci établi, il a fallu définir le fonctionnement du script puis commencer le développement.

Concernant le fonctionnement, nous avons établi ce schéma fonctionnel :



Actuellement, toutes les fonctions spécifiées sur le schéma fonctionnel sont développées ainsi qu'une fonction utilitaire.

Le fonctionnement de chaque fonction est décrit ci-dessous ainsi que la description des différentes limitations techniques et problèmes rencontrés.

Fonction CheckHardware :

Ce fut la première fonction développée, cette fonction est simplement une cascade d'exécution de commande UNIX permettant de récupérer les différentes composantes d'un système. Les composantes ciblées sont donc le nombre de cœurs, la quantité de mémoire vive, une approximation du débit ascendant et descendant, la distribution du système d'exploitation et sa version. Elle va ensuite afficher à l'utilisateur le résultat.

Dans le cadre du développement du script il était important de savoir si la machine dispose d'une connectivité suffisante pour satisfaire les besoins d'un serveur web. Toutefois il est difficile d'établir un modèle permettant de récupérer une variable reflétant dans sa globalité la connectivité d'une machine avec l'entièreté d'Internet. Nous avons donc dû choisir une solution réalisable, en conséquence nous mesurons le débit entre la machine cliente et un serveur mis à disposition par la société GitHub. Dans l'éventualité où l'URL d'accès à ce serveur changerait nous avons placé cette variable au début du code afin de pouvoir la modifier aisément.
Cependant, ce serveur retourne des valeurs au format Human Readable. Cela pose un problème pour le traitement de ces données. En effet le format Human Readable traduit les octets en fonction de leur puissance, par exemple MegaOctets, GigaOctets …

Fonction HRtranslation :

Ainsi pour palier à ce problème il a fallu développer la fonction HRtranslation qui permet de convertir un débit au format Human Readable en octets par seconde.
Son fonctionnement est simple, on définit un dictionnaire associant une chaîne de caractères à une puissance. Ensuite on divise la chaîne de caractères donnée en argument en deux chaînes de caractères. La première contient la valeur du débit, l'autre l'unité.
Enfin la fonction renvoie la valeur du débit multiplié par la valeur associée à la chaîne de caractères de l'unité.

Fonction CheckSoftware :

Le script devant, à terme, gérer Nginx et Apache, il fallait détecter quels logiciels sont actifs sur la machine. Cette fonction va encore une fois exécuter une suite de commandes UNIX et de conditions pour déterminer si Apache et/ou Nginx sont exécutés par la machine.

Fonction CheckNGINXFiles :

Ensuite le script doit récupérer la configuration afin de vérifier l'état de chaque paramètre surveillé. Cette fonction vérifie donc que le fichier de configuration de Nginx existe et n'est pas vide.

De plus elle va détecter et isoler une partie des blocs de configurations de tous les sites pris en charge par Nginx. Cela va permettre d'émettre des recommandations pour chaque site hébergé par la machine, ce qui facilite grandement l'exploitation des données par les utilisateurs.

Fonction GetNGINXParam :

Cette fonction permet donc de parcourir les blocs de configurations isolés auparavant et d'en extraire dans un format utilisable le nom et la valeur de chaque variable que l'on surveille.

Afin de maximiser l'efficacité de la fonction nous avons défini un dictionnaire contenant les noms de toutes les variables à récupérer. Au moyen d'une boucle for in nous pouvons récupérer les valeurs.

La problématique majeure de cette fonction était le stockage du couple nom et valeur. La solution choisie initialement était de créer une variable du nom du paramètre, ayant pour valeur la valeur du paramètre. Cette manipulation n'est malheureusement plus possible avec python 3.
Nous avons donc choisi d'utiliser un fichier JSON* pour stocker chaque couple de valeur.
Par la suite, ce système d'indexation nous permettra de traiter ces conditions via une cascade de conditions.

6) Résultats :

Cela conclut les avancements réalisés sur ce projet. Bien que le développement ne soit pas terminé, la base du script est présente. De plus, le projet repose sur une documentation complète et des expérimentations qui permettent de valider son bien-fondé ainsi que le gain minimal de performances atteignable.

Retour d’expérience :

Ce stage de dix semaines réalisé au sein de Jaguar Network m’aura été grandement bénéfique tant sur le côté technique que sur le plan humain.

Cela m’aura permis de découvrir l’activité au sein d’un support technique, l’impact que cette activité a sur les différents environnements de production de tous les clients de la société. Cela m’aura aussi permis de développer ma capacité à communiquer, aussi bien avec les clients qu’avec le reste des équipes techniques.

De plus, ce stage m’a permis de me plonger dans l’écosystème Jaguar Network, qui est très dense. L’apprentissage des différents outils et technologies utilisés à la fois pour les besoins internes que pour ceux des clients m’a permis d’élargir ma vision de cette industrie.

Concernant la réalisation des scripts m’ayant été confiés, cela m’aura permis de faire de grand progrès tant en administration système qu’en développement. Cela m’aura aussi permis de découvrir les méthodes et les bonnes pratiques du développement en entreprise. Cela s’est matérialisé par la production d’une documentation détaillant les différentes spécificités techniques de ces projets. Mais aussi par l’utilisation d’outils tel qu’un logiciel de contrôle de version du code source. Tout ceci a permis de renforcer ma volonté de poursuivre dans le développement orienté infrastructure.

Pour conclure, ce stage a renforcé mon intérêt pour le domaine informatique et réseau, et enrichit mes connaissances au travers d’expériences vécues.

Remerciements :

Le stage en entreprise effectué au sein de Jaguar Network est ma première expérience professionnelle dans le milieu des Réseaux et Télécommunications.

Je voudrais remercier mon maître de stage, Yoann NORMAND, pour m'avoir fait confiance, donné des responsabilités pendant la durée du stage ainsi que pour tous ses conseils.

Je tiens aussi à remercier Pierre FOREST, Nicolas POLIZZI et Guillaume MOURIES, pour leur accueil, également pour leurs nombreux conseils, leur écoute face à mes questions, et surtout leurs réponses et leurs aides.

Je remercie également tout le personnel de Jaguar Network et notamment l'équipe du support et du déploiement pour m'avoir accueilli et intégré dans l'entreprise.

Glossaire :

File Descriptor : Une file descriptor est un nombre qui définit un fichier ouvert par le système d'exploitation de la machine.

Accès I/O : Les accès I/O sont les accès que le système d'exploitation fait sur la mémoire physique de la machine, cela lui permet de récupérer des valeurs contenues sur le disque. Ces accès sont faits, soit pour les besoins propres du système d'exploitation, soit afin de faire fonctionner les applications exécutées par le système d'exploitation. Ainsi une latence lors d'un accès I/O peut nuire à la rapidité d'exécution d'une application.

Virtual Host : Le mécanisme d'hébergement virtuel permet d'héberger plusieurs sites web différents sur une même machine, ainsi chaque virtual host sera hébergé avec une configuration dédiée.

CMS : Content Management System, cela désigne un type d'applications permettant de créer et de mettre à jour aisément des sites web. C'est un système de gestion de contenu.

JSON : JavaScript Object Notation est un format de données permettant d'associer un couple de valeur. Son utilisation est originellement destinée à gérer des objets du langage de programmation JavaScript.

Point de montage : Le point de montage est le répertoire du système de fichier à partir duquel les données d'un autre périphérique, système de fichier, partition ou disque dur sont accessibles

Etat « deleted » : Un fichier est en état deleted lorsqu'il est supprimé mais qu'il continue d'être utilisé, ainsi le fichier n'apparait pas dans la liste des fichiers mais son volume continue d'augmenter

Sitographie :

Nginx, 'Rick Nelson', 2014 [consulté en Avril 2019], Tuning NGINX for Performance, via https://www.nginx.com/blog/tuning-nginx/

Nginx, 'Alan Murphy, 2019 [consulté en Avril 2019], Performance Tuning – Tips & Tricks, via https://www.nginx.com/blog/performance-tuning-tips-tricks/

Nginx, [consulté en Avril 2019], Compression and Decompression, via https://docs.nginx.com/nginx/admin-guide/web-server/compression/

DBA-Oracle, 'Donald Burleson', [consulté en Avril 2019], What is the best webserver hardware configuration?, via http://www.dba-oracle.com/t_best_webserver_hardware_configuration.htm

Bjornjohansen, 'Bjørn Johansen', 2014 [consulté en Mai 2019], Optimizing HTTPS on Nginx, via https://bjornjohansen.no/optimizing-https-nginx

Github, 'Denji', 2015 [consulté en Avril 2019], NGINX Tuning For Best Performance, via https://gist.github.com/denji/8359866

Gagor, 'Timor', 2016 [consulté en Avril 2019], Optimize Nginx for performance, via https://gagor.pl/2016/01/optimize-nginx-for-performance/

Testdriven, 'Michael Herman', 2019 [consulté en Avril 2019], Developing an Asynchronous Task Queue in Python, via https://testdriven.io/blog/developing-an-asynchronous-task-queue-in-python/

TuneTheWeb, 'Barry Pollard', 2016 [consulté en Avril 2019], OPINION - HTTP versus HTTPS versus HTTP/2, via https://www.tunetheweb.com/blog/http-versus-https-versus-http2/

Kinsta,2019 [consulté en Mai 2019], How to Speed up Your WordPress Site (Ultimate 2019 Guide), via https://kinsta.com/learn/speed-up-wordpress/#

Annexes :

Code source du projet du recommandation Apache et Nginx :

```
#Debut du script
###
# Zone imports
###
import subprocess #Allows to execute shell commands
import os #Allows to check path
import json
###
# Zone variable
###
units = {"bit/s": 1, "Kbit/s": 10**3, "Mbit/s": 10**6, "Gbit/s": 10**9, "Tbit/s":
10**12}
###
# Zone dev
###
#Allows to translate Human Readable to bytes
def HRtranslation(size):
    number, unit = [string.strip() for string in size.split()]
    return int(float(number)*units[unit])

def CheckHardware():
       #Vcore detection
       vcore_cmd="lscpu | grep 'CPU(s)' |awk '{print $2}'| head -n 1"
       vcore_raw = subprocess.Popen(vcore_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
       vcore_str=vcore_raw.stdout.read().decode('utf-8')
       #Ram detection
       ram_cmd="cat /proc/meminfo | grep 'MemTotal' | awk '{print $2}'"
       ram_raw = subprocess.Popen(ram_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
       ram_str=ram_raw.stdout.read().decode('utf-8')

       #Get download bandwidth
       bandwidth_download_cmd="curl -s
https://raw.githubusercontent.com/sivel/speedtest-cli/master/speedtest.py | python -
| grep Download | awk '{print $2,$3}'"
       bandwidth_download_raw = subprocess.Popen(bandwidth_download_cmd,
stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
       bandwidth_download_str=bandwidth_download_raw.stdout.read()
    bandwidth_download=HRtranslation(bandwidth_download_str.decode('utf-8'))
       #Get upload bandwidth
       bandwidth_upload_cmd="curl -s
https://raw.githubusercontent.com/sivel/speedtest-cli/master/speedtest.py | python -
| grep Upload | awk '{print $2,$3}'"
       bandwidth_upload_raw = subprocess.Popen(bandwidth_upload_cmd,
stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
       bandwidth_upload_str=bandwidth_upload_raw.stdout.read()
    bandwidth_upload=HRtranslation(bandwidth_upload_str.decode('utf-8'))
    #Get OS
    OS_cmd="cat /etc/*-release | grep '^ID=' | cut -d '=' -f2"
       OS_raw = subprocess.Popen(OS_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
       OS_str=OS_raw.stdout.read().decode('utf-8')
       #Get version
    OS_version_cmd='cat /etc/*-release | grep "VERSION_ID" | grep -o "....$" | cut -
c 1-3'
```

```
	OS_version_raw = subprocess.Popen(OS_version_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
	OS_version_str=OS_version_raw.stdout.read().decode('utf-8')
    #Format output
    print("Votre configuration actuelle \n")
    print("Nombre de coeurs : "+str(vcore_str)+"\n")
	print("RAM : "+str(ram_str)+"\n")
	print("OS : "+str(OS_str)+"\n")
	print("Version : "+str(OS_version_str)+"\n")
	print("Approximation de la bande passante descendante :
"+str(bandwidth_download)+"\n")
	print("Approximation de la bande passante montante :
"+str(bandwidth_upload)+"\n")

def CheckSoftware():

	#Nginx detection
	#((ps aux | grep -E 'nginx: worker process' | grep -v color) && echo $?) |
tail -n 1
	nginx_cmd="((ps aux | grep -E 'nginx: worker process' | grep -v color) &&
echo $?) | tail -n 1"
	nginx_raw = subprocess.Popen(nginx_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
	nginx_str=nginx_raw.stdout.read().decode('utf-8')[:-1]
	if nginx_str == "0":
		nginx_str="Running"
	else:
		nginx_str="Stopped"
	#Apache2 detection
	apache_cmd="((ps aux | grep -E 'apache2' | grep -v color) && echo $?) | tail
-n 1"
	apache_raw = subprocess.Popen(apache_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
	apache_str=apache_raw.stdout.read().decode('utf-8')[:-1]
	if apache_str == "0":
		apache_str="Running"
	else:
		apache_str="Stopped"
	print("Nginx state : "+str(nginx_str))
	print("Nginx state : "+str(apache_str))
	return

def CheckNGINXFiles():

	#Find if nginx.conf exists and is not empty
	nginx_conf_cmd="ls -al /etc/nginx/nginx.conf | awk '{print $5}'"
	nginx_conf_raw = subprocess.Popen(nginx_conf_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
	nginx_conf_str=nginx_conf_raw.stdout.read().decode('utf-8')[:-1]
	if nginx_conf_str != "0" and nginx_conf_str.isdigit():
		nginx_conf_str="Exists"
	else:
		nginx_conf_str="Does not exists!"
	nginx_conf_block_cmd="nginx -T 2>/dev/null | grep 'server .*{\|server_name
\|listen \|access_log \|error_log ' | grep -v '#' | tr '\n' ' '"
	nginx_conf_block_raw = subprocess.Popen(nginx_conf_block_cmd,
stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
	nginx_conf_block_str=nginx_conf_block_raw.stdout.read().decode('utf-8')
	conf_blocks=nginx_conf_block_str.split("server {")
	conf_blocks=list(filter(None, conf_blocks)) #Allows to filter empty strings
in the list
	print("Tous les blocks : |"+str(nginx_conf_block_str)+"|")
	print("Blocs séparés : |"+str(conf_blocks)+"|")
```

```
def GetNGINXParam():

	json_string='{'
	standard_format_variable = ["worker_connections ", "multi_accept ",
"reset_timedout_connection ","send_timeout ","sendfile ","keepalive_timeout
","ssl_session_cache ","ssl_session_timeout ","ssl_prefer_server_ciphers
","ssl_ciphers ","ssl_stapling ","ssl_stapling_verify ","gzip ","gzip_disable
","gzip_vary ","gzip_proxied ","gzip_comp_level ","gzip_buffers
","gzip_http_version"]
	special_format_variable = ["ssl_protocols ","add_header  ","gzip_types "]
	global_variable=standard_format_variable+special_format_variable
	for i in global_variable:
		if i in standard_format_variable:
			loop_cmd="nginx -T 2>/dev/null | grep -v '#' | grep '"+i+"' |
awk '{print $2}'| cut -d  ';' -f1"
			loop_raw = subprocess.Popen(loop_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
			loop_str=loop_raw.stdout.read().decode('utf-8')[:-1]
			print("Valeur loop str : |"+loop_str+"| "+str(i)+"|")
			json_string=json_string+'"'+str(i)[:-1]+'":
"'+str(loop_str)+'", '
		elif i in special_format_variable:
			loop_special_cmd="nginx -T 2>/dev/null | grep -v '#' | sed -n
'/"+i+"/,/;/p'"
			loop_special_raw = subprocess.Popen(loop_special_cmd,
stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
			loop_special_str=loop_special_raw.stdout.read().decode('utf-
8')[:-1]
			print("Valeur loop str : |"+loop_str+"| "+str(i)+"|")
			json_string=json_string+'"'+str(i)[:-1]+'":
"'+str(loop_special_str)+'", '
	json_string=json_string[:-2]+'}'
	with open('json_string_fome.txt', 'w') as json_file:
		json.dump(json_string, json_file)
	#Get use_epoll value
	use_epoll_cmd="grep -o 'use epoll;' /etc/nginx/nginx.conf"
	use_epoll_raw = subprocess.Popen(use_epoll_cmd, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
	use_epoll_str=use_epoll_raw.stdout.read().decode('utf-8')[:-1]
CheckHardware()
CheckSoftware()
CheckNGINXFiles()
GetNGINXParam()
```

Code source du script analysant les partitions :

```
## check_part ##
#!/bin/bash

#Check_part version 3
#Written by Lucas Ripoche for Jaguar Network (Marseille) on Monday, April 15, 2019

#Run as root for protected partitions

check_part () { #This function will find the most filled partition
	#Allows to select the right partiton
	#Important : First tail will remove the headline, second is necessary due to
an unknown error (first and last line are switched), thanks to the second tail we can
remove the first line matching the least filled partiton
	part=$(df -h -x devtmpfs -x tmpfs -l | grep -v boot | rev | awk '{print $1"
"$2}' | rev | tail -n +2 | sort -n | tail -1)
```

```
	#Allows to relate the partition to the mounting point
	mount_point=$(echo $part | awk '{ print $2}')
	echo "La partition la plus utilisée est : $part"
}

analyse () { #This function allows to find the 4 largest files from the selected mounting point
	#Allows to check the number of files/directories of the mounting point (k allows to have this number in bytes)
	nb_inode=$(df -ik $mount_point | awk '{print $3}' | tail -n +2)
	#If the number of files/directories is greater than 1 million --> stop the script
	if [ $nb_inode -gt 1000000 ]; then
		echo "Nombre de fichiers trop important"
		exit 0
	fi
	#Checking if there is any file in "deleted" state (deleted file but pointer still active)
	deleted=$(lsof $mount_point | grep 'deleted')
	#If last command has succeeded then there are deleted-state files
	if [ "$?" -eq 0 ]; then
		echo 'Des fichiers sont en état deleted'
		echo "$deleted"
	fi
	#Allows to list and display the heaviest files in the selected directory (doens't count less than 10Mo files)
	info=$(find $mount_point -xdev -type f -size +100000k -exec ls -alh {} \; | awk '{print $5"\t"$9}' |sort -rh| head )
	echo "$info"
}

if [ "$#" -eq 1 ]; then #Checking the number of arguments, if there is 1, the first argument is the partition to check (manual mode)
	part=$1

	mount_point=$(df | grep $part | rev | grep -v 'Mounted on' | awk '{ print $1}' | rev)
	echo "La partition analysée est : $part"
	echo "Cette partition correspond au dossier : $mount_point"
	analyse $mount_point
elif [ "$#" -eq 0 ]; then #Else it will select the most filled partition (auto mode)
	check_part
	analyse
else #Else, incorrect number of arguments
	echo "Nombre d'arguments incorrect "
fi
```

Code source du sciprt analysant les fichiers de journalisation :

```
## analyseur_log ##
#!/bin/sh
#analyser_log version 2
#Written by Lucas Ripoche for Jaguar Network (Marseille) on Wednesday, April 17, 2019
file_exists () { #This function will check if the file exists
    test -f $path #Test if the path is a file
    if [[ "$?" == 1 ]]; then #If the command hasn't succeeded, then stop the script
       echo "Le fichier n'existe pas"
    exit 0
fi
}
echo "Chemin du fichier ?"
read path
file_exists
echo "Choisissez le type d'analyse : "
echo "1. Top 10 des IPs ayant fait le plus de requête "
```

```
echo "2. Nombre de requête par heure"
echo "3. Nombre de requête par jour"
echo "Veuillez rentrer le numéro de l'option voulue"
read analyse
if [[ analyse -eq 1 ]]; then #Top 10 case
    if [[ "$path" =~ \.gz$ ]]; then #Checking if the path is finished by a .gz
        zresult=$(zcat $path | awk '{print $1}' | sort |uniq -c |sort -n| tail | tac)
#Allows to find and count every occurencies of the ip in the compressed access log
       echo "$zresult"
       else
       result=$(awk '{print $1}' $path | sort |uniq -c |sort -n| tail | tac) #Allows
to find and count every occurencies of the ip in the access log
       echo "$result"
    fi
elif [[ analyse -eq 2 ]]; then #Hour case
       echo "Quel jour voulez vous analyser (Exemple 23 Juin 2019, entrez 23/Jun/2019)
?"
       read date
       echo "Quelle heure voulez vous analyser (Exemple 11h, entrez 11) ?"
       read hour
    if zgrep -q "$date" "$path" #Checking if the date is contained in the file
    then
       if [[ "$path" =~ \.gz$ ]]; then #Checking if the path is finished by a .gz
            zresult=$(zgrep "$date"":""$hour" "$path" | cut -d[ -f2 | cut -d] -f1 |
awk -F: '{print $2":"$3}' | sort -nk1 -nk2 | uniq -c | awk '{ if ($1 > 10) print $0}')
#Allows to find and count every occurencies of the selected hour and each minutes of
it
             echo "$zresult"
        else
        result=$(grep "$date"":""$hour" "$path" | cut -d[ -f2 | cut -d] -f1 | awk -F:
'{print $2":"$3}' | sort -nk1 -nk2 | uniq -c | awk '{ if ($1 > 10) print $0}') #Allows
to find and count every occurencies of the selected hour and each minutes of it
        echo "$result"
    fi
    else
             echo "Le fichier de log ne contient pas la date mentionnée"
       fi
elif [[ analyse -eq 3 ]]; then #Date case
       echo "Quel jour voulez vous analyser (Exemple 23 Juin 2019, entrez 23/Jun/2019)
?"
       read date
    if zgrep -q "$date" "$path" #Checking if the date is contained in the file
    then
       if [[ "$path" =~ \.gz$ ]]; then #Checking if the path is finished by a .gz

            zresult=$(zgrep "$date" "$path" | cut -d[ -f2 | cut -d] -f1 | awk -F:
'{print $2":00"}' | sort -n | uniq -c) #Allows to find and count every occurencies of
the selected date and each hours of it
             echo "$zresult"
        else
             result=$(grep "$date" "$path" | cut -d[ -f2 | cut -d] -f1 | awk -F:
'{print $2":00"}' | sort -n | uniq -c) #Allows to find and count every occurencies of
the selected date and each hours of it
             echo "$result"
        fi
    else
        echo "Le fichier de log ne contient pas la date mentionnée"
    fi
else
       echo "Veuillez rentrer un nombre entre 1 et 3"
fi
```