UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

DÉPARTEMENT D'INFORMATIQUE ET D'INGÉNIERIE

DÉVELOPPEMENT D'UN RÉSEAU DÉFINI PAR LOGICIEL (SDN) PROGRAMMABLE, TRANSPARENT ET OUVERT

MÉMOIRE PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DU PROGRAMME DE MAÎTRISE EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION

PAR

YVON ZAFIMAHEFA ANDRIANIRINA

UNIVERSITE DU QUEBEC EN OUTAOUAIS

DEPARTEMENT D'INFORMATIQUE ET D'INGENIERIE

DÉVELOPPEMENT D'UN RÉSEAU DÉFINI PAR LOGICIEL (SDN) PROGRAMMABLE, TRANSPARENT ET OUVERT

Par

Yvon Zafimahefa Andrianirina

Pour l'obtention du grade de maîtrise es science (m.sc.)
En sciences et technologies de l'information

Jury d'évaluation

Prof. Luigi Logrippo ……. Président du Jury

Prof. Omar Abdul-Wahab …… Membre du Jury

Prof. Larbi Talbi…… Directeur de Recherche

Dédicace

À Dieu tout puissant, qui m'a donné la grâce, la volonté, la persévérance et les moyens pour la réalisation de ce mémoire de maîtrise.

À ma femme Andoniaina Ramanarivo et mes deux enfants Yvannah et Yvan Andrianirina pour leur patience et leur soutien durant mes études.

À mon oncle Abdon Razafimahefa qui m'a beaucoup aidé et soutenu dans ce travail et pendant mes études ainsi qu'à sa famille pour leur soutien.

À mes parents, de près ou de loin, qui m'ont porté dans leurs prières et qui m'ont soutenu tout au long de mes études.

À mes frères et à ma sœur.

À la communauté Malagasy d'Ottawa-Gatineau.

Remerciements

Je voudrais exprimer ma gratitude à mon directeur de recherche Professeur Larbi Talbi, pour son aide financière, ses conseils tout au long de ce projet de recherche et pour sa patience.

Mes remerciements s'adressent aussi :

- À Mr Jamal Hadi Salim de la compagnie MOJATATU pour son aide dans ce projet, ainsi qu'à ses collaborateurs, notamment, Dr. Evangelos Haleplidis.
- À Mr Stephen Jaworski, de la même compagnie, pour ses conseils techniques.
- Aux membres du Jury, Professeur Luigi Logrippo et Professeur Omar Abdul-Wahab qui n'ont épargné aucun effort pour l'amélioration du contenu de ce mémoire.

Table des matières

Liste des figures

Liste des acronymes

- API : Application Programing Interface
- OAS : Open API Specification
- SDN : Software Defined Network
- IOT : Internet Of Things
- TSDB : Time-Series Data-Base
- PoD : Point Of Deployment
- PdD : Point de Deployment
- ToR : Top of Rack
- OS : Operating System
- Rx : Réception
- Tx : Transmission
- NE : Network Element
- FE : Forwarding Element
- CE : Control Element
- FEID : Forwarding Element Identification
- JSON : JavaScript Object Notation
- YAML : Yet another Market Language

Résumé

Un nouveau paradigme du réseautage informatique, appelé *Software Defined Network* (SDN), a vu le jour récemment et attire l'attention de plusieurs géants de l'informatique et des télécommunications.

Ce concept est basé sur la séparation du plan de données (*data plane*) responsable des transferts de paquets et du plan de contrôle (*Control plane*) responsable de la prise de décision, qui sont traditionnellement liés. Il est clair que ce concept apporte une grande amélioration de performances sur l'implémentation et la vue globale du réseau, ainsi que sur la prise de décision du trafic.

L'idée principale de ce mémoire est de trouver une solution pour améliorer la programmabilité du SDN, en insérant une nouvelle couche, afin de permettre aux administrateurs de développer ou de modifier eux-mêmes leurs applications de contrôle et de gestion du réseau. Cette solution aura l'avantage d'être orientée objet et ouverte.

Nous testerons notre approche en simulant les plans de données et de contrôle et en développant nous-même des applications de gestion du réseau, dont l'ensemble constituera notre SDN.

Abstract

A new paradigm of computer networking called Software Defined Network (SDN) has recently emerged and attracted the attention of several IT and telecommunications giants.

This concept is based on the separation of the data plane responsible for packet transfers, and the Control plane responsible for decision making, which are traditionally linked. It's clear that this concept will increase the performance of the network implementation or the overall view and as well on traffic decision making.

The principal idea of this thesis is to bring a solution to improve the programmability of SDN, by inserting a new layer in order to allow network administrators to develop or modify their own network control and management applications. This solution will have the advantage to be open and object oriented.

We will test our approach by simulating data and control planes then developing network management applications ourselves, all of which will constitute our SDN.

Chapitre 1

Identification du problème et motivation

Face aux innovations technologiques, comme l'infonuage (*Cloud Computing*), l'internet des objets (IoT – *Internet of Things*) ou la virtualisation des machines et des systèmes, on se rend compte que leurs mis en réseaux deviennent de plus en plus problématique. En effet, l'implémentation traditionnelle est particulièrement complexe à configurer à cause de la nature distribuée des protocoles de réseaux, d'une part, et par la dépendance du plan de contrôle avec le plan de données des équipements interconnectés, d'autre part.

Le paradigme SDN (*Software Defined Networking*) est une nouvelle approche qui préconise une architecture où tout le plan de contrôle du réseau est logiquement centralisé dans un composant détaché du plan de données. Ainsi, la configuration d'un réseau revient alors à programmer ce composant; appelé contrôleur SDN [1].
Actuellement, il existe plusieurs versions et types de contrôleur SDN qui sont déjà disponibles. Ces contrôleurs adoptent des interfaces ou protocoles de communications spéciaux, pour permettre les communications avec les applications de gestion du réseau. Présentement, ces interfaces sont encore limitées, notamment dues aux faits qu'elles soient spécifiques et propriétaires [2], aussi, développées de façon bas niveau [3].

L'objectif du travail présenté dans ce mémoire consiste donc à insérer une nouvelle couche, dans l'architecture du SDN, afin, d'une part, de permettre la transparence et l'accessibilité des services disponibles dans le contrôleur SDN, et d'autre part, de donner la possibilité aux utilisateurs et aux administrateurs de réseaux de développer ou d'améliorer leurs applications de contrôle selon leurs besoins, de façon modulaire, ouverte et orientée objet.
Cette approche est l'une des premières à se focaliser sur l'interface Nord de SDN et d'aborder cette lacune qui fait des SDN actuels, partiellement programmables.
Dans cette optique, elle pourrait amener à une normalisation de ces protocoles dans le concept SDN.

Nous commençons d’abord par présenter ce qu’est SDN pour mieux identifier la problématique. Ensuite, nous présenterons notre solution et ses points forts, ainsi que les technologies connexes dont nous avons besoin pour développer notre SDN.

Nous enchainerons avec la présentation de notre méthodologie qui inclura la partie de mise en pratique, c’est-à-dire, le développement de notre SDN afin d’avoir des résultats concrets. Et enfin, nous évaluerons les résultats et présenterons notre conclusion et les perspectives futures.

Chapitre 2

Définition et problématique du SDN

Constituant l'état de l'art de ce travail, on explique dans ce chapitre les principes de base et l'architecture de SDN. Cela nous aidera à mieux cerner la problématique dont fera l'objet de notre apport de solution.

2.1 Principe du SDN

Selon E. Haleplidis et ses confrères [4]: la mise en réseau définie par logiciel SDN est un terme du paradigme des réseaux programmables qui fait référence à la capacité des applications logicielles à programmer, de façon dynamique et individuelle, les périphériques d'un réseau. Ceci permet de contrôler le comportement du réseau dans son ensemble. Boucadair et Jacquenet [5] soulignent que SDN est un ensemble de techniques utilisées pour faciliter la conception, la livraison, et l'exploitation des services de réseau d'une manière déterministe, dynamique et évolutive.

Un élément clé du SDN est l'introduction d'une abstraction entre le plan de transmission (plan de données) et le plan de contrôle, qui sont traditionnellement liés, dont leurs rôles sont les suivants : le plan de transmission transfère le trafic en fonction des décisions prises par le plan de contrôle, qui a son tour, décide comment gérer le trafic du réseau. Cette abstraction permet de séparer et de fournir aux applications les moyens nécessaires pour contrôler le réseau, par programme ou par requête. L'objectif est de tirer avantage de cette séparation, afin de réduire la complexité et de permettre une innovation plus rapide sur les deux plans [6]. Donc une centralisation logique du contrôle du réseau, qui facilitera sa gestion et son évolution.

L'historique sur l'évolution du domaine de la recherche et du développement du SDN est évoqué par Nick Feamster, Jennifer Rexford et Ellen Zegura [7] : en commençant par l'amélioration des protocoles, et après par la virtualisation des réseaux et, finalement, le paradigme SDN.

2.2 Architecture globale du SDN

Une architecture de réseau basée sur SDN divise les processus (configuration, l'allocation des ressources, hiérarchisation du trafic, et redirection du trafic dans le matériel sous-jacent) en trois couches de base : la couche d'application, la couche de contrôle (plan de contrôle), et la couche infrastructure (plan de données). Chacune des couches a les caractéristiques suivantes : une limite bien définie, un rôle spécifique, et des interfaces programmables d'applications (API [8]) pertinentes pour communiquer avec les couches adjacentes. Une comparaison, entre le modèle traditionnel de contrôle du trafic distribué dans les périphériques et l'architecture SDN centralisée, est illustrée par la figure suivante :

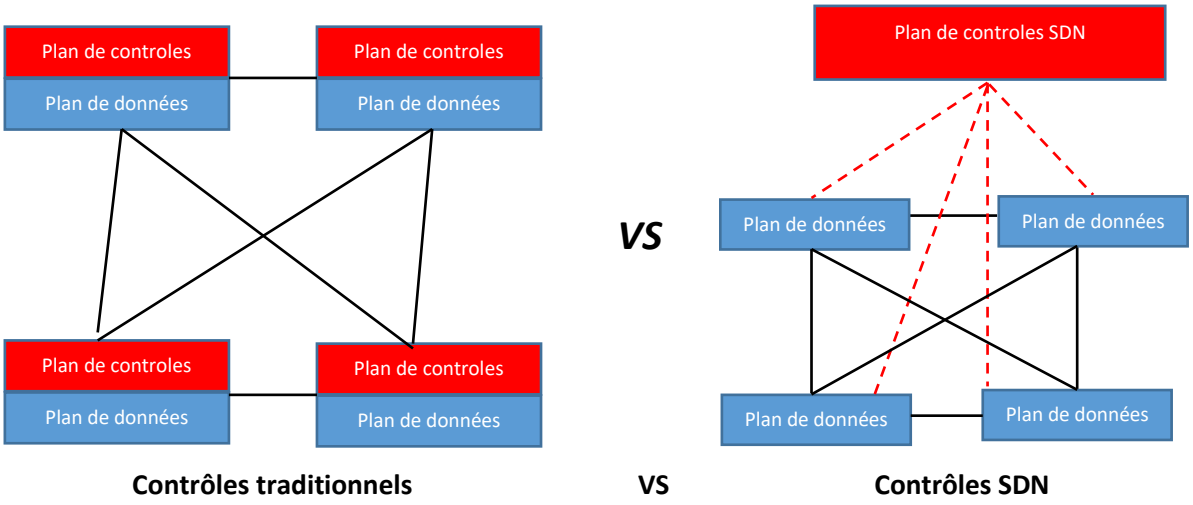

Figure 1 : Comparaison entre les contrôles traditionnels et centralisés d'un SDN

L'architecture standard d'un SDN est illustrée par la figure suivante:

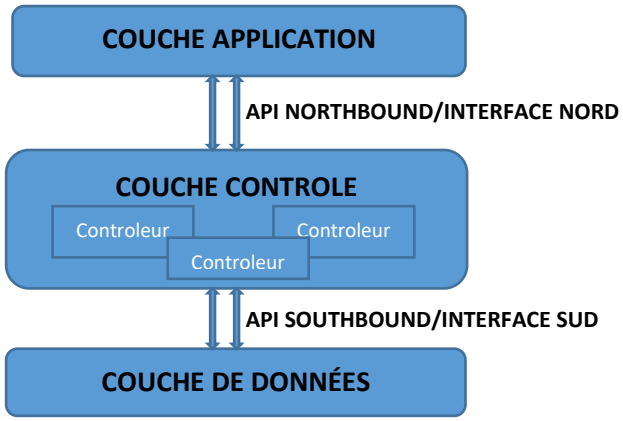

Figure 2 : Architecture Standard d'un SDN.

2.2.1 La couche infrastructure ou le plan de données (transfert):

Un plan de données est défini par un ensemble de composants de réseaux, tels que des commutateurs, des routeurs, des équipements virtuels etc. L'objectif d'un plan de données est de transmettre le trafic réseau sur une base d'un certain ensemble de règles de transmission ordonnée par le plan de contrôle. La communication entre le plan de données et le plan de contrôle est assurée par des interfaces Sud qu'on appelle *API Southbond*.

2.2.2 La couche contrôle ou le plan de contrôle:

Le plan de contrôle est responsable de la prise de décision du trafic à travers le réseau, dépendamment des critères suivants: en fonction des exigences de l'application, selon les utilisateurs, et des politiques de communication du réseau résultant du plan de données. Le composant central d'un plan de contrôle est le contrôleur SDN. Un contrôleur SDN traduit les exigences des applications et les objectifs commerciaux tels que la nécessité de hiérarchiser le trafic (qualité de service), le contrôle d'accès (privilège), la gestion de la bande passante, ou autre. Ensuite, ces informations sont communiquées aux composants du plan de données.

Selon la taille du réseau, il peut y avoir plusieurs contrôleurs SDN. La configuration du réseau, via le plan de contrôle, donne la possibilité de manipuler en temps réel les tableaux de flux de chaque élément du réseau, en fonction des performances du réseau et des exigences du service. Le contrôleur donne une vue claire et centralisée du réseau sous-jacent, ce qui donne un outil de gestion du réseau flexible et puissant, qui peut affiner les règles de performance.

2.2.3 La couche d'application ou le plan d'application:

Le plan d'application contient des programmes spécifiques au réseau et des logiciels commerciaux. Une vue abstraite du réseau sous-jacent est présentée aux applications via des interfaces Nord (*Northbound API*). Le niveau d'abstraction peut inclure des paramètres du réseau tels que les descripteurs de retard/de débit, et la disponibilité des ressources.

Les applications demandent la connectivité entre les nœuds en tenant compte des paramètres exigés. Une fois que le chemin (*path*) optimal est défini, le contrôleur SDN configure, individuellement, les éléments du réseau dans le plan de données.

*2.2.4 Les interfaces Sud (*Southbond API*)*

Les interfaces Sud constituent un protocole entre le contrôleur SDN et le plan de données. Elles contrôlent les opérations de transfert, les notifications d'événements, les rapports statistiques et annoncent également les capacités du réseau. Essentiellement, elles permettent à un contrôleur de définir le comportement du matériel dans le réseau. Des types d'interfaces Sud sont connus comme OpenFlow [9] ou ForCES [10].

*2.2.5 Les interfaces Nord (*Northbound API*)*

Les interfaces Nord représentent une abstraction de fonctions réseaux avec une interface programmable. En d'autres termes, elles permettent aux applications de consommer les services du réseau, et dynamiquement, changer leur comportement.

L'architecture et les interfaces Nord des contrôleurs SDN varient selon les fournisseurs. Certains vendeurs incorporent des contrôleurs SDN dans leurs applications, tandis que d'autres définissent des interfaces Nord pour faciliter le protocole de communication entre les contrôleurs et leurs propres services SDN au niveau de la couche application.

Ces aspects actuels des interfaces Nord ont attiré notre attention et nous ont conduits à déterminer les problématiques suivantes:

2.3 Problématiques

A titre de rappel, il faut savoir que pour développer des applications de gestion d'un SDN, les développeurs ou les administrateurs de réseaux se réfèrent aux services disponibles via l'interface Nord. Idéalement, celle-ci doit fournir d'une part, une abstraction permettant de cacher toutes les interactions entre les modules de contrôle et l'infrastructure physique, telles

que, les commandes pour configurer les infrastructures du réseau, les requêtes pour les statistiques, les informations sur la topologie physique du réseau, etc. et d'autre part, des spécifications des interfaces de programmation, qui sont expressives et accessibles à tous.
L'administrateur peut ainsi développer, selon ses besoins, des fonctions et des modules de contrôles indépendants qui représenteront ses principaux services réseaux (contrôle d'accès, transport, équilibrage de charge, etc...).

Une interface Nord expressive, modulaire et flexible, est donc une condition essentielle pour réaliser les objectifs premiers du paradigme SDN. Malheureusement, d'après Messaoud Aouadj, dans sa thèse en doctorat [11], ou de V. Tijare et D. VaSudevan [3], les interfaces Nord actuelles n'offrent que très peu de possibilités de modularités, et de réutilisabilités. Entre autres, les APIs existantes (de bas niveau) sont dépourvues de concept qui permet de regrouper des actions dans un même conteneur logique pour réaliser une même tâche de plus haut-niveau.

Selon D. Kreutz et al. [2], les interfaces Nord actuelles sont propriétaires, spécifiques, et prédéfinies. En effet, si les administrateurs ou les clients veulent améliorer leurs applications de gestion du réseau, ou s'ils veulent créer leurs propres applications, ils devront faire appels aux fournisseurs, ce qui entrainera un coup supplémentaire en temps et en argent. En d'autres termes, si les clients veulent de nouvelles fonctionnalités, ils seront obligés d'envoyer des requêtes aux fournisseurs du SDN. Le traitement des requêtes du client suit un très long processus, à savoir, l'attente d'une offre de faisabilité et du développement suivant le rythme du fournisseur. Dans certains cas, les résultats ne sont pas satisfaisantes alors on doit refaire le même processus jusqu'à la satisfaction du client et par conséquent le coût est évident en temps et en espèce.
Par contre, si le SDN est ouvert au niveau des interfaces Nord, les administrateurs peuvent ajouter ou améliorer des fonctionnalités, telles que, le partage de charges dynamique (*load balancing*) ou la priorisation de trafic selon le profil des utilisateurs donnés et selon leurs besoins.

Notre travail est alors de proposer l'insertion d'une nouvelle couche au-dessus des interfaces Nord existantes s'appuyant sur le travail de Farzaneh Pakzad et al. [12] sur l'amélioration de la topologie standard. Cette couche exposera les services sous forme d'un langage de plus haut niveau, donnant une possibilité de programmer les applications en « Orienté Objet ».

Tenant compte de l'état de l'art sur le paradigme SDN, notre approche devra apporter plus de flexibilité au niveau des interfaces Nord (modulaire, réutilisation, et expressive). Par conséquent, cette nouvelle couche doit avoir les propriétés suivantes:

- ✓ Assurer l'expressivité, la modularité, et la flexibilité des programmes de contrôle,
- ✓ permettre de spécifier les politiques décrivant les services réseaux,
- ✓ permettre de développer les applications nécessaires pour la gestion et la configuration du réseau de façon modulaire, orientée objet, et ouvert.

Chapitre 3

Objectif principal

Dans ce chapitre, compte tenu des défis sus-cités par rapport à l'interface Nord, nous proposons l'insertion d'une nouvelle couche qui nous permettra de rendre les interfaces Nords sous forme de langage de programmation de haut niveau (orienté objet) et de façon ouverte.

Cette couche va nous permettre de spécifier les services de l'interface Nord, de les rendre accessibles, modulaires et réutilisables.

Nous allons tout d'abord illustrer notre idée, ensuite, nous allons expliquer comment cette couche pourrait être la solution de notre problématique.

3.1 Création de la couche OAS

La figure suivante illustre le projet d'insertion de la couche appelée OAS -*Open API Spécification*- [13].

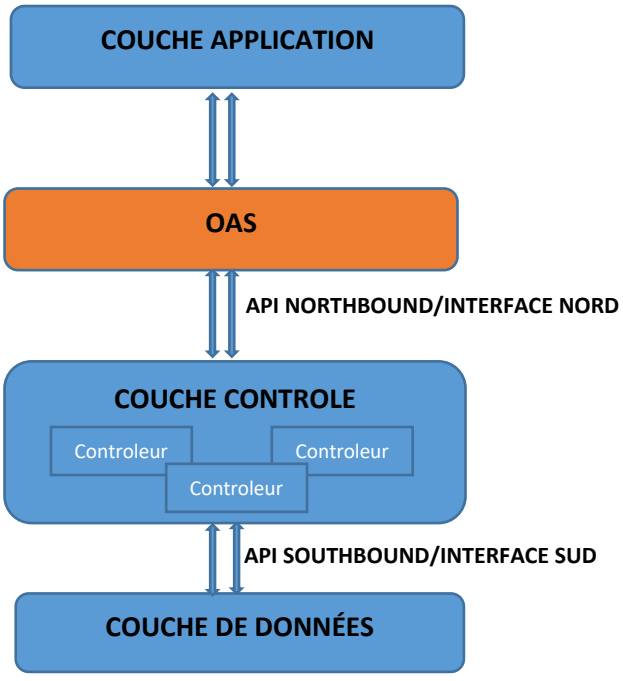

Figure 3: Architecture SDN standard avec la nouvelle couche OAS

Comme nous le voyons sur la figure 3, OAS se placera au niveau de l'interface Nord et il sera l'interface qui nous permettra de développer notre couche application.

3.2 Propriétés de l'OAS

La connaissance des propriétés d'OAS nous aidera à comprendre ses capacités. En effet, d'après Erlin Mc Kean [14], l'OAS est un standard indépendant du langage de programmation. Il permet aux utilisateurs de découvrir les services et de comprendre leurs capacités sans accéder au code source ou à la documentation.

Il n'est pas nécessaire de lier un logiciel à un service, en d'autres mots, un service n'appartient pas forcément au créateur de sa description. Cependant, il faut que les capacités des services soient décrites dans la structure de l'OAS. Il faut noter que le format d'écriture de l'OAS est en YAML [15], ou en JSON [16].

L'une des avantages de la spécification OAS est la possibilité, avec des éditeurs comme SWAGGER[1], de générer une bibliothèque d'objets qui sont définis par les services décrits. Ces objets permettent ensuite de programmer des applications de façon orientée objets. Des exemples d'utilisations des objets sont disponibles dans cette bibliothèque.

Notons que OAS est un concept ouvert au public et entièrement gratuit.

Nous pourrons donc tirer avantage de ces propriétés pour pouvoir :

- Spécifier les services disponibles dans les interfaces Nord existants
- Générer des objets afin de développer les applications de gestion des réseaux de façon haut niveau et orienté objet

3.3 Rôles de l'OAS

En se basant sur l'explication de Ron Ratovsky [17], l'OAS peut apporter aux interfaces programmables les améliorations suivantes :

- ✓ la réutilisabilité améliorée des services offerts par l'API existante,

[1] http://editor.swagger.io.

- ✓ la possibilité de changement dynamique des paramètres,
- ✓ une meilleure gestion des contenus,
- ✓ les relations entre les objets,
- ✓ la clarté des exemples, et
- ✓ une définition de sécurité améliorée.

Par conséquent, les rôles de l'OAS dans le SDN sont les suivants :

- Facilite la programmation d'un SDN, via une interface ouverte.
- Permet d'accéder aux fonctionnalités définies et publiées par les fournisseurs de services.

Nous pensons que l'insertion de l'OAS dans l'architecture SDN est une bonne solution parce que, non seulement elle permet aux fournisseurs de SDN de préserver leurs codes sources, mais elle donne aussi aux utilisateurs/clients la possibilité de programmer ouvertement les applications selon leurs besoins et de contrôler dynamiquement l'ensemble de leurs réseaux.

3.4 Structure de l'OAS

Dans ce travail, nous utilisons la dernière version d'OAS qui est la 3.0 et la figure suivante nous montre la structure de l'OAS 3.0 [18]

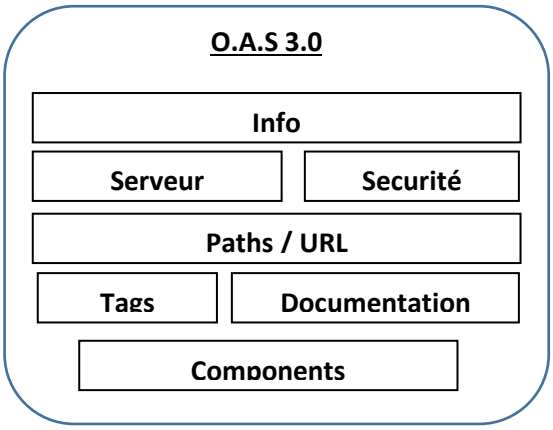

Figure 4: Structure d'O.A.S 3.0

Explication :

Info :

- Information sur la version de l'OAS
- Information globale sur les services à spécifier
- Information sur le concepteur

Serveur et sécurité :

- Déclaration du serveur hôte des services.
- Précision sur le niveau de sécurité requise.

Paths / URL :

- Adresse des différents services disponibles.
- Notification des variables.

Tags et documentation :

- Spécification des services
- Spécification des paramètres
- Spécification des métadonnées
- Spécification des valeurs d'entrées et de sorties
- Spécification des états des requêtes.

Components :

- Spécification des contenus des données de sorties

3.5 Prérequis pour l'insertion d'OAS

La figure suivante nous montre les étapes de la spécification avec OAS.

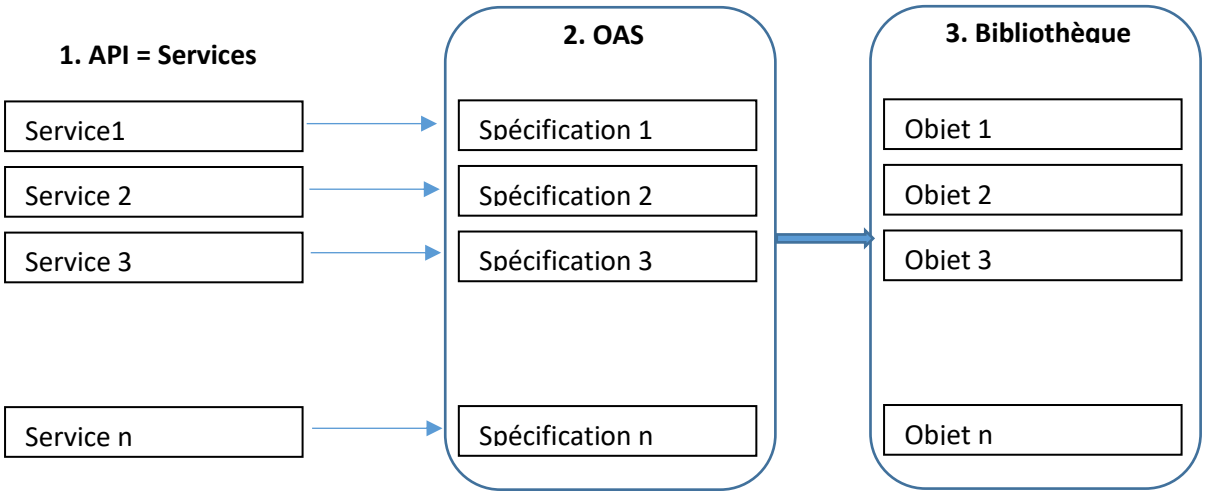

Figure 5: Prise en main d'OAS

Explication :

1. Connaître toutes les API / Services
2. Spécifier chaque service et les regrouper dans un OAS
3. Générer la bibliothèque d'objets pour le développement d'applications

Remarque :

- Il est primordial dans un projet de développement d'OAS d'avoir accès à tous les services disponibles dans l'interface Nord et de les comprendre. Toutefois, un tel prérequis n'est pas évident avec les existants actuels, car le domaine SDN est encore innovant et tous les concepteurs veulent conserver leurs propriétés.
- Ayant acquis cette étape, les spécifications doivent être explicites et bien détaillées afin de faciliter leurs réutilisations et leurs compréhensions. Aussi, chaque service connu doit être spécifié pour que la couche soit complète.
- L'OAS développé dans ce travail serait une documentation typique de ce que pourrait être les spécifications standard des interfaces nord du SDN.
- OAS est beaucoup utilisé actuellement dans les applications de gestion de bases de données pour des sites commerciaux, et donc nous serons l'une des premières à l'insérer dans un environnement de réseaux informatique.

Chapitre 4

Outils de simulation du SDN

Afin de tester l'efficacité de notre approche, il nous faut créer un environnement SDN, et pour se faire, il nous faut constituer les trois (03) couches constituantes d'un SDN.

Notre but serait d'accéder à l'interface Nord, de spécifier les services disponibles et d'en générer des objets qui nous servent à développer les applications de gestion du réseau.

Ce chapitre décrit trois (03) outils qui nous permettront de simuler les couches infrastructure et contrôle d'un SDN et aussi, de constituer une partie de la couche application.

4.1 ARACHNE

ARACHNE [19] est une application de simulation d'un centre de données, c'est-à-dire que cet outil nous permet de simuler à la fois la couche infrastructure et la couche de contrôle d'un SDN.

Nous optons à utiliser ARACHNE pour deux raisons: la première, est qu'il est développé par la compagnie MOJATATU [20] avec laquelle nous avons eu une collaboration étroite pour la réalisation de ce projet et dont le support a été assuré par leurs ressources. La deuxième est que pour l'implémenter, il suffit d'utiliser un ordinateur doté d'un système d'exploitation basé sur linux.

Il est possible de visualiser et de configurer de façon dynamique la topologie d'un réseau donné (serveurs, commutateurs, autres éléments).

Notons qu'un serveur intégré dans ARCHNE appelé MOJASWAG, est l'équivalent de l'interface Nord proposé par MOJATATU. Tandis que, le protocole FORCES assure l'API Sud de l'environnement.

Plus de détails sur cet environ est disponible dans l'annexe 1.

4.2 PROMETHEUS

Nous avons choisi PROMETHEUS [21] qui est un serveur de données de types temps/séries pour faire partie de notre couche application. À noter que ce serveur est gratuit et compatible avec plusieurs systèmes d'exploitation. Ce module est une application à concept ouvert et se développe dans un environnement communautaire (on peut avoir de l'aide facilement, et l'évolution du module est toujours mise à jour).

Ce serveur récupère les données dans des serveurs cibles et les stocke dans sa base de données de type temps/séries (*Time Series Data Base –TSDB-*).

Dans ce projet, PROMETHEUS nous permet de récupérer et de stocker les données de types temps/séries fournies par le réseau simulé par ARACHNE, comme par exemple, les activités des ports.

Par la suite, en tant que serveur, il expose ces données métriques pour être récupérer par des applications d'analyse de données et de visualisation graphique.

Les détails sur ce serveur sont disponibles dans l'annexe 2.

4.3 GRAFANA

Encore une fois, le module GRAFANA [22] est un concept ouvert qui permet d'analyser graphiquement les données, notamment comme celles exposées par PROMETHEUS.

On a choisi ce module qui fait aussi partie de la couche d'applications, car il possède multiples fonctionnalités de visualisation en temps réel et d'analyse des métriques provenant des éléments du réseau comme les statistiques des interfaces (nombre de paquets entrant/sortant, erreurs, etc.).

Dans ce projet, GRAFANA est notre interpréteur graphique des données stockées dans PROMETHEUS, elle nous permet également d'analyser ces données

L'annexe 3 nous donne plus de détails sur GRAFANA.

Chapitre 5

Méthodologie suivie pour l'insertion de OAS

Pour atteindre nos objectifs c'est à dire prouver que, grâce à l'insertion d'OAS, les services seront spécifiés de manière claire et le développement de la couche d'application d'un SDN se fait de façon haut niveau et ouvert, nous proposons la méthodologie dont les étapes sont les suivantes :

- Développer OAS pour spécifier les services disponibles sur MOJASWAG,
- Programmer, de façon modulaire et Orienté Objet, les exportateurs de métriques faisant partie de notre couche d'application SDN,
- Evaluer l'efficacité d'OAS dans un SDN.

5.1 Développement de l'OAS

Pour pouvoir écrire et prendre en main l'OAS, il nous faut les démarches suivantes:

- Prendre connaissance et comprendre les services disponibles sur MOJASWAG.
- Ajouter/développer le nouveau code OAS avec un éditeur spécifique, et nous optons sur le logiciel libre appelé SWAGGER, qui est disponible à l'adresse suivante : http://editor.swagger.io.
- L'éditeur SWAGGER nous permet de développer les spécifications des services de MOJASWAG et de générer une bibliothèque d'objets de type client/serveur. Ces objets sont écrits selon le langage désiré, et dans notre cas, nous optons pour le PYTHON [23] version 2.7.17.

Remarque :

Comme évoqué dans l'introduction, les SDN existants sont propriétaires et fermés et de ce fait, connaitre les services disponibles de l'interface Nord existant serait un challenge. Heureusement, notre collaboration avec la compagnie MOJATATU nous a permis d'accéder à des services clés pour valider notre concept. À titre d'exemple, l'utilisation de l'interface Nord MOJASWAG dans ARACHNE.

5.2 Développement d'un environnement SDN

Pour pouvoir évaluer l'efficacité d'OAS, il nous faut à ce stade, développer un environnement SDN. Dans ce projet, nous simulons d'abord les couches infrastructure et contrôle avec ARACHNE, ensuite, nous développons des exportateurs de métriques qui font partie de la couche application à l'aide de notre bibliothèque d'objets, et enfin, nous utilisons PROMETHEUS et GRAFANA pour les analyses et interprétations graphiques.

5.3 Evaluation d'OAS dans un SDN

Dans cette partie, l'évaluation d'OAS est basée sur les éléments suivants :

a) La clarté de la documentation.
b) La fiabilité des objets générés par l'OAS via SWAGGER.
c) Les temps de réponses des applications développées via ces objets.
d) Les erreurs possibles dans le développement d'OAS.
e) L'intégration d'OAS dans un SDN, c'est-à-dire son efficacité dans l'ensemble du SDN.

Chapitre 6

Résultats préliminaires

Ce chapitre fait l'objet de la mise en pratique de notre approche. On présentera des résultats préliminaires obtenus de la validation de notre approche. Ces résultats sont un bon augure vers l'approfondissement de notre approche. Dans un premier temps, nous illustrons par la figure 6 sa structure globale, ensuite, nous expliquons étape par étape les processus dans le développement de notre SDN.

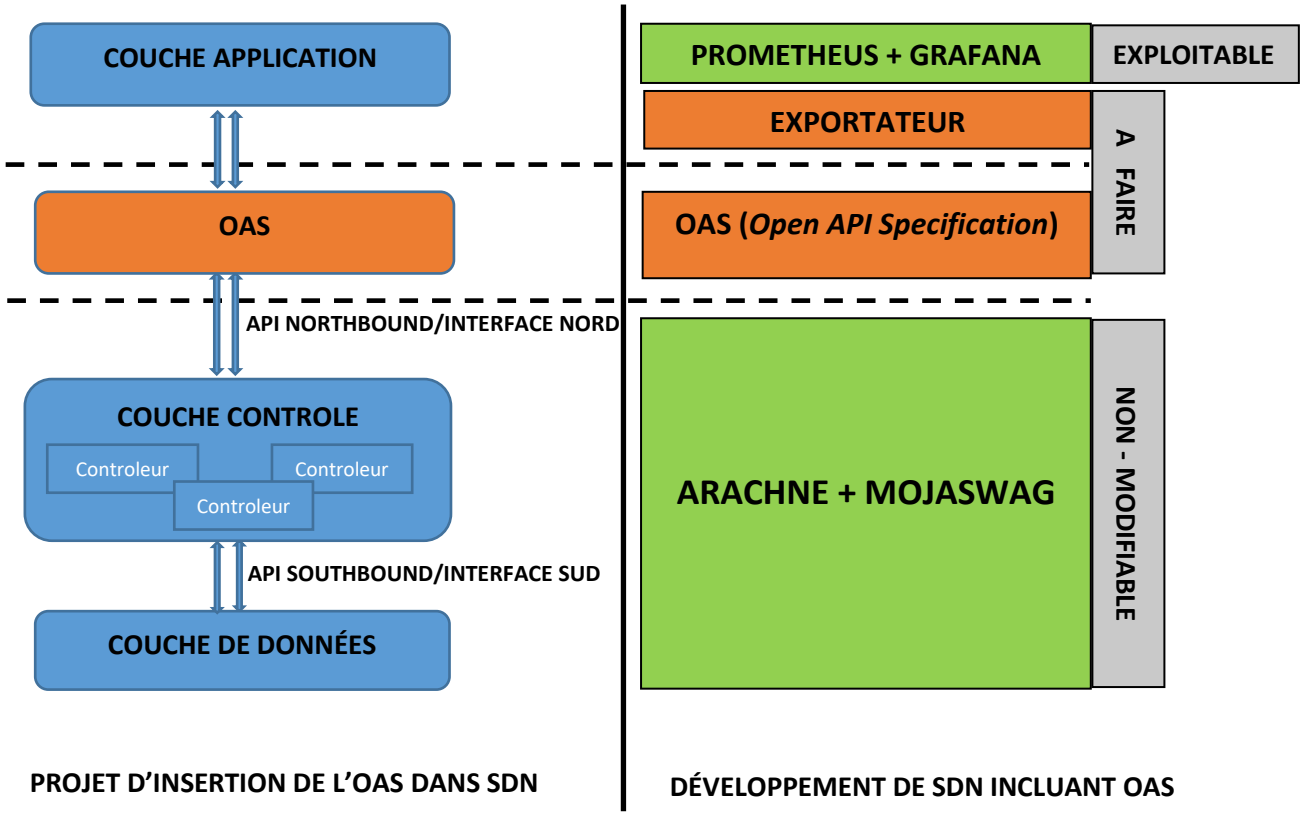

Figure 6: Structure générale du projet

Evoqué dans le chapitre précèdent, ARACHNE sera notre outils de simulation à la fois de la couche données et de la couche contrôle dont l'interface nord est MOJASWAG.
Il ne sera pas modifiable mais configurable selon la topologie de réseau voulue.
Ensuite, Nous allons développer nous-même notre code OAS et les applications d'exportation de données et de configuration de paramètres

Enfin, nous allons paramétrer PROMETHEUS et GRAFANA pour l'exploitation des données.

6.1 Simulation des couches infrastructure et contrôle

6.1.1 Travailler dans un environnement Linux

- ✓ Implémentation et configuration de L'OS (*Operating System*) qui est Ubuntu 18.04 lts
- ✓ Familiarisation avec les différentes lignes de commandes Linux : pour les installations, les configurations, les lancements d'applications, etc.

6.1.2 Simulation avec ARACHNE

- ✓ Implémentation et configuration d'ARACHNE

```
yvon@Asus:~$ sudo chmod 777 /dev/kvm
[sudo] password for yvon:
yvon@Asus:~$ cd arachne
yvon@Asus:~/arachne$ ./arachne -p ./arachne_plugins_forces/
Need root to creating arachne0 and arachne1 interface for you.
Waiting for that VM is booted up...
Linux ArachneVM 4.16.8 #1 SMP 2018-03-01 x86_64 GNU/Linux
done

    ___                          __                    __
   /   |  _____________  ______/ /_  ____  ___       | /  \ |
  / /| | / ___/ __  / ___/ __ \/ __ \/ _ \     \_\\  //_/
 / ___ |/ /  / /_/ / /__/ / / / / / /  __/       //()\\
/_/  |_/_/   \__,_/\___/_/ /_/_/ /_/\___/        \\  //

Welcome to arachne version 0.1
```

Figure 7: ARACHNE avec le protocole ForCEs

- ✓ Simulation d'un réseau clos composé de :
  - Quatre (04) racks (L1_R1_P1_Z1,…)

    ```
    arachne % set -r 4
    ```

  - Trois (03) serveurs (H3_R1_P1_Z1,…) pour chaque rack.

    ```
    arachne % set -se 3
    ```

  - Deux (02) commutateurs (S1_P1_Z1, S2_P2_Z2) pour interconnecter les 4 racks.

```
arachne % set -sp 2
```

Remarque :

On a du se limiter par rapports aux nombres d'éléments du réseau pour diverses raisons :

- Limitation de ressources (puissance de notre ordinateur) car plus les éléments simulés sont nombreux plus ceux-là demandent plus de puissance.
- La vitesse d'exécution des requêtes est trop lente à partir de nombre de ports de 150.
- Les envois de paquets avec la commande « pingall » présentent beaucoup de pertes de paquets quand les éléments (ports) sont nombreux.

6.1.2.1 Comprendre l'architecture du réseau.

La figure suivante illustre la topologie de notre réseau clos simulé par ARACHNE

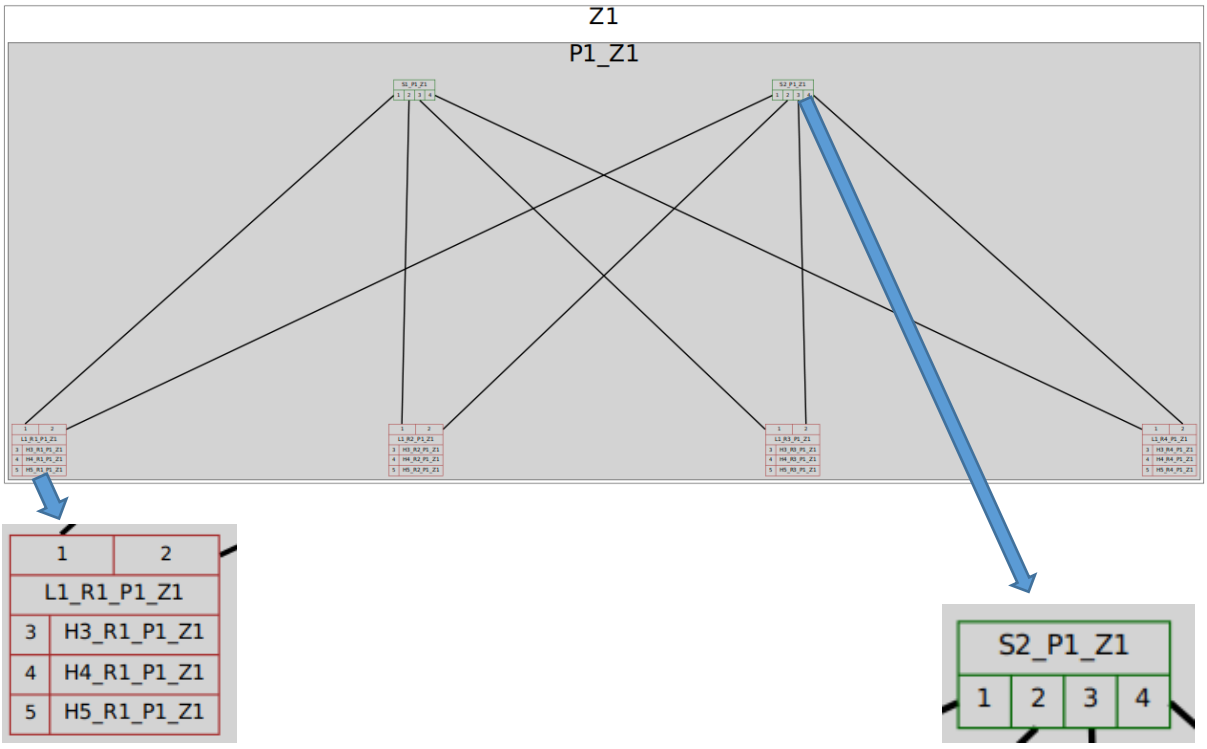

Figure 8: Topologie du réseau Clos simulé par ARACHNE

6.2 Développement de l'OAS

6.2.1 Comprendre les services de MOJASWAG et les données qu'ils exposent :

Dans notre travail, on se concentre sur trois (03) services que nous jugeons suffisants pour mettre à l'épreuve notre approche.
Les figurent 9, 10, 11 suivantes illustrent ces services ainsi que ses données de réponses.

6.2.1.1 Service d'inventaire d'éléments du réseau (Forwarding Elements ou FE)

Localhost :8000/NE/FE

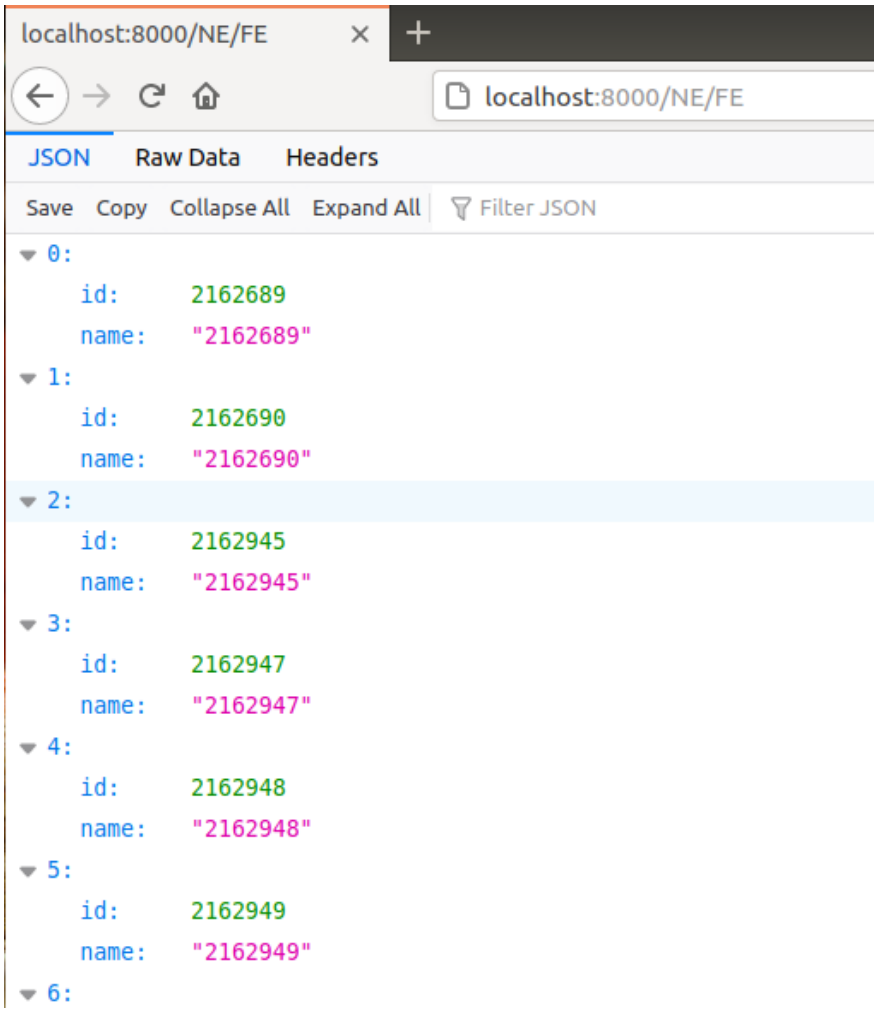

Figure 9: Liste des éléments du réseau : FE (Forwarding Elements)

6.2.1.2 *Services de recueil d'activités des ports de façon granulaire pour chaque FE.*

Localhost: 8000/FE/{num_FEId}/Port/1/ports

Localhost: 8000/FE/{num_FEId}/Port/1/ports/{num_port}

Figure 10: Statistiques d'un port spécifique d'un élément du réseau

6.2.1.3 *Service de configuration de recueil et de configuration des paramètres FOOBAR*

Localhost: 8000/FE/num_FEId/MyLFB/1/foobar

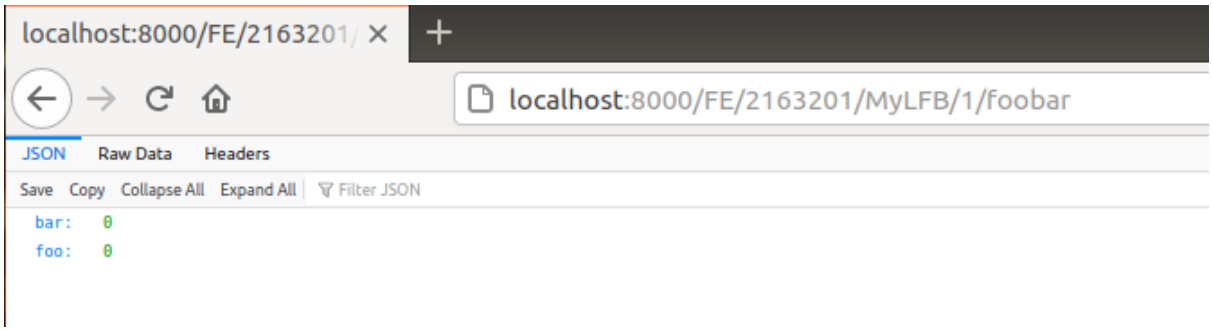

Figure 11: Service de paramétrage FOO/BAR

6.2.2 Écrire OAS et générer les objets pour chaque service.

6.2.2.1 Familiarisation avec SWAGGER.

Nous avons écrit avec SWAGGER un code OAS pour quelques services du MOJASWAG et ensuite, nous avons généré la bibliothèque d'objets SWAGGER_CLIENT. Notons que chaque objet de la bibliothèque est associé à chaque service.
Nous avons développé OAS en YAML.
Nous présenterons les étapes de développement d'OAS comme suit :
Après chaque figure de spécification de service, nous présenterons l'objet y afférant dans la bibliothèque.

Les figures suivantes montrent les différentes parties de notre OAS.

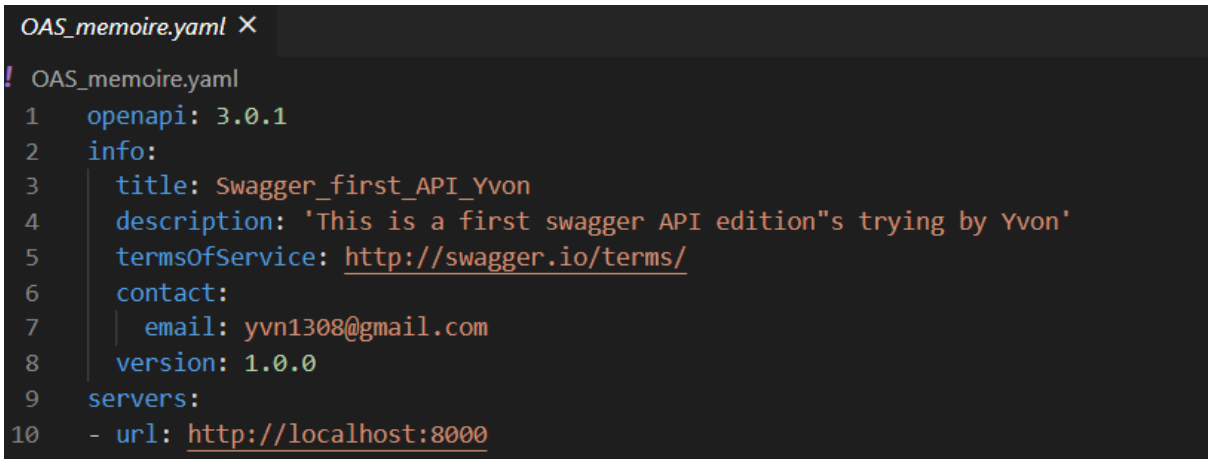

Figure 12: Code OAS (entête).

Un OAS doit contenir un entête qui spécifie sa version ainsi que le profil de son créateur et l'adresse du serveur de tous les services à spécifier.

Explication :

Il s'agit de l'OAS version 3.0.1

- Titre : Premier OAS d'Yvon
- Description : Première version d'OAS de Yvon
- Adresse du serveur de services : 127.0.0.1:8000 ou localhost :8000

6.2.2.2 Spécifications des services

Après avoir pris connaissance des services disponibles et accessibles, ou du moins dont on aura besoin, nous pouvons commencer les spécifications de ces derniers avec OAS.

Nous allons donc spécifier un par un les services suscités :

- ✓ **Service d'inventaire d'éléments du réseau (Forwarding Elements ou FE)**

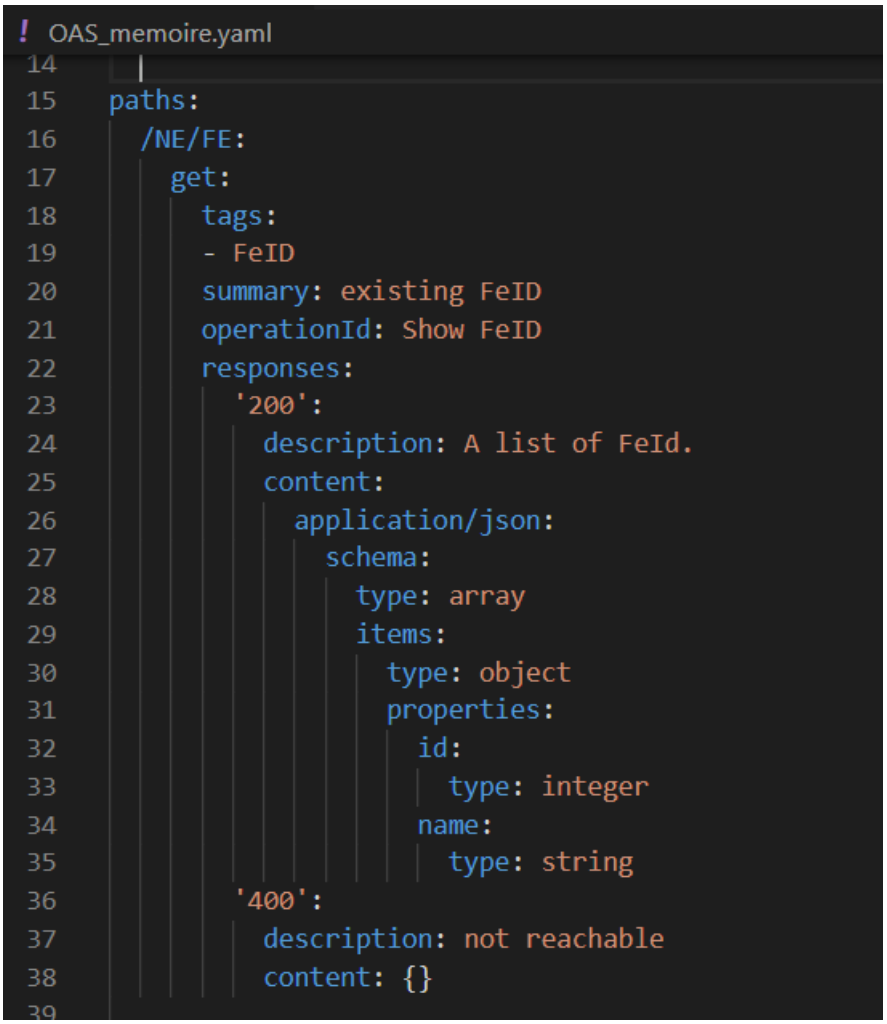

Figure 13: OAS NE_FE

Explication :

L'API: serveur/NE/FE

- Avec la requête http GET, nous récupérons tous les éléments existants dans le réseau clos (datacenter)

- Les valeurs de sorties seront de type liste ou tableau qui auront une Identité (Id) de type entier et un nom (*name*) de type caractère.
- Le statut 200 nos indique la réussite de la requête tandis que le statut 400 nous indique le contraire.

L'objet généré dans la bibliothèque est : *SWAGGER_CLIENT.FEIDAPI ()*

✓ **Services de recueil d'activités des ports de façon granulaire pour chaque FE.**

OAS_memoire.yaml

```yaml
/FE/{FeID}/Port/1/ports:
  get:
    tags:
    - FEID / List of ports and stats on specific FE
    summary: List of ports on specific FE
    description: show all of ports on a specific FE
    operationId: getFEID
    parameters:
    - name: FeID
      in: path
      description: ID of FE return
      required: true
      schema:
        type: integer
        format: int64
    responses:
      200:
        description: All of FEID Ports
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/feid'
```

Figure 14: OAS_ Liste des ports dans un FE

Explication :

L'API : serveur/FE/FEID}/Port/1/ports

- Avec la requête http GET, nous récupérons la liste de tous les ports existants.
- {FEID} : C'est l'identité, de type entier, d'un élément du réseau et l'accolade {} nous indique que c'est la variable ou argument (paramètre) de l'objet.

- Le statut 200 indique la réussite de la requête et donne en retour la liste des ports et toutes les activités. Ceci est spécifié par : $ref : '#/components//schemas/feid'

L'objet généré est : *SWAGGER_CLIENT.FEIDPORTSANDACTIVITYAPI ()*

✓ **Services de recueil d'activités des ports de facon granulaire pour chaque FE.**

OAS_memoire.yaml

```yaml

/FE/{FeID}/Port/1/ports/{PortID}:
  get:
    tags:
    - FeID specific ports activity
    summary: Find Actity by FeID
    description: Return a single port's activity
    operationId: getFeIDById
    parameters:
    - name: FeID
      in: path
      description: ID of FeID return
      required: true
      schema:
        type: integer
        format: int64
    - name: PortID
      in: path
      description: ID of Port return
      required: true
      schema:
        type: integer
        format: int64
    responses:
      200:
        description: FeiD specific port Activity
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/feid'

      400:
        description: not reachable
        content: {}

```

Figure 15: OAS_ Recueil activité dans un port spécifique

Explication :

L'API : server/FE/{FEID}/Port/1/ports/{PortID}

- Avec la requête http GET, nous recueillons les activités d'un port spécifique.
- Les paramètres {FEID} et {PortID} représentent respectivement l'identité de l'élément du réseau et le numéro du port spécifique y figurant.
- Le statut 200 évoque l'aboutissement de la requête et retourne toutes les activités d'un port spécifié.

L'objet généré est : *SWAGGER_CLIENT.FEIDSPECIFICPORTSACTVITYAPI ()*

- ✓ **Service de configuration de recueil et de configuration des paramètres FOO/BAR**

```
! OAS_memoire.yaml
 96
 97      /FE/{FeID}/MyLFB/1/foobar:
 98        get:
 99          tags:
100          - foo values
101          summary: Show the foo and bar value
102          description: show foo and bar value
103          operationId: foovalue
104          parameters:
105          - name: FeID
106            in: path
107            description: ID of FeID return
108            required: true
109            schema:
110              type: integer
111              format: int64
112          responses:
113            200:
114              description: FeiD foo value
115              content:
116                application/json:
117                  schema:
118                    $ref: '#/components/schemas/foobar'
```

Figure 16: OAS Recueil des valeurs des paramètres FOO/BAR

Explication :

L'API: serveur/FE/{FeID}/MyLFB/1/foobar

- Avec la requête http GET, cette API nous permet de récupérer les valeurs des deux paramètres FOO et BAR.
- Notons que ces paramètres sont généralement des paramètres de validation de concept utilisés par les concepteurs afin de prouver que le concept est fonctionnel. Dans notre cas, *FOO* et *BAR* sont des paramètres configurables qui représenteront les restes de tous ces types de paramètres de chaque port.
- Nous aurons besoin de manipuler ces deux paramètres afin de montrer par la suite qu'avec OAS on peut aussi atteindre à toutes les configurations ou paramétrages d'un SDN.

L'objet généré dans la bibliothèque est : SWAGGER_CLIENT.FOOVALUESAPI ()

✓ **Service de configuration du paramètre FOO**

OAS_memoire.yaml

```
/FE/{FeID}/MyLFB/1/foobar/foo:
  put:
    tags:
    - foo values
    summary: update the foo value
    description: updating foo value
    operationId: updatefoo
    parameters:
    - name: FeID
      in: path
      description: ID of FeID return
      required: true
      schema:
        type: integer
        format: int64
    requestBody:
      description: update the foo value
      content:
        text/plain:
          schema:
            type: string
      required: true
      x-codegen-request-body-name: body

    responses:
      200:
        description: foo value Updated
        content: {}
```

Figure 17: OAS paramétrage de FOO

Explication :

L'API serveur/FE/{FEID}/MyLFB/1/foobar/foo

- Avec la requête GET : Cette API nous permet de recueillir la valeur de FOO d'un port
- Avec la requête PUT : Cette API nous permet de configurer la valeur de FOO d'un port spécifié.
- On remarque que seule l'identité FEID d'un élément du réseau est le paramètre à définir pour accéder à FOO et de le configurer.

L'objet généré pour configurer FOO est :
SWAGGER_CLIENT.FOOVALUESAPI.UPDATEFOO ()

✓ **Service de configuration du paramètre BAR**

OAS_memoire.yaml

```yaml
/FE/{FeID}/MyLFB/1/foobar/bar:
  put:
    tags:
    - bar values
    summary: update the bar value
    description: updating bar value
    operationId: updatebar
    parameters:
    - name: FeID
      in: path
      description: ID of FeID return
      required: true
      schema:
        type: integer
        format: int64
    requestBody:
      description: update bar value
      content:
        text/plain:
          schema:
            type: string
      required: true
      x-codegen-request-body-name: body

    responses:
      200:
        description: bar value Updated
        content: {}
```

Figure 18: OAS_ paramétrage valeur de BAR

Explication:

L'API serveur/FE/{FEID}/MyLFB/1/foobar/bar

- Avec la requête GET : Cette API nous permet de recueillir la valeur de BAR d'un port
- Avec la requête PUT : Cette API nous permet de configurer la valeur de BAR d'un port spécifié.
- On remarque que seule l'identité FEID d'un élément du réseau est le paramètre à définir pour accéder à BAR et de le configurer.

L'objet généré dans la bibliothèque pour configurer BAR est :
SWAGGER_CLIENT.BARVALUESAPI.UPDATEBAR ()

6.3 Développement de la couche application

6.3.1 Développer l'application exportatrice de métriques par intégration d'OAS

Dans ce projet nous prouvons qu'OAS est efficace dans le développement des applications de gestion du SDN autant dans la récupération des données, que dans les configurations du réseau. Dans un premier temps nous développons des applications de récupération des données et nous développons ensuite une application de configuration typique.

6.3.1.1 Développer l'exportateur pour récupération des métriques.

La figure suivante montre un exportateur de métriques. Nous pouvons constater par la suite que le développement s'est fait de façon Orientée Objet en utilisant notre bibliothèque d'objets SWAGGER_CLIENT.
Ces applications nous permettent de récupérer :

- La liste des éléments du réseau
- Le nombre de ports de chaque élément
- La liste des valeurs de FOO/BAR de tous les éléments du réseau

```
1  from __future__ import print_function
2  import swagger_client
3  import time
4  import swagger_client
5  import json
6  from swagger_client.rest import ApiException
7  from pprint import pprint
8  from prometheus_client import Gauge,CollectorRegistry,pushadd_to_gateway,push_to_gateway
9  registry = CollectorRegistry()
10 registry1 = CollectorRegistry()
11 registry2 = CollectorRegistry()
12 registry3 = CollectorRegistry()
13 registry4 = CollectorRegistry()
14
15 #configuration = swagger_client.Configuration()
16 #configuration.host = "http://localhost:8000"
17
18 api_instance = swagger_client.FeIDApi(swagger_client.ApiClient())
19 api_instance1 = swagger_client.FeIDSpecificPortsActivityApi(swagger_client.ApiClient())
20 api_instance2 = swagger_client.FeIDPortsAndActivityApi(swagger_client.ApiClient())
21
22 duration = Gauge('Yvonjob_duration_seconds', 'Duration of collect job', registry=registry4)
23
24
```

Figure 19: Instanciation des objets dans l'exportateur des listes des ports

Explication :

- Ligne 1-8 : Importation de toutes les bibliothèques nécessaires
- Ligne 9-13 : Définition de tous les registres dont on aura besoin
- Ligne 18-20 : Instanciation des objets générés auparavant.

La figure 20 nous montre les codes de recueil de la liste des ports avec leurs activités

```python
try:
    with duration.time():
    # existing FeID
        api_response = api_instance.show_fe_id()
        x = len(api_response)
        print("there are {} Fe available".format(x))
        element = x
        nombre = 0
        for i in range (0,x):
            #Liste des Elements avec leur ID
            print(" FeID {} = {} ".format(i+1, api_response[i].id))
            x=api_response[i].id
            api_response2 = api_instance2.get_fe_id_ports(x)
            # Nombre total des ports dans le reseau clos
            y = len(api_response2)
            nombre = nombre + y
            FE = Gauge("FE_ID{}".format(i), "List FeID", registry=registry)
            FE.set(x)
            pushadd_to_gateway('localhost:9091', job='FEID_LIST', registry=registry)
            Port = Gauge("Number_of_Port_on_FE_{}".format(x), "Number of Port", registry=registry1)
            Port.set(y)
            pushadd_to_gateway('localhost:9091', job='NUMBER_PORT', registry=registry1)

            # Exporter les RX de chaque port de chaque element
            for j in range (0,y):
                print("j={}".format(j))
                z = api_response2[j].value.stats.rx_packets
                Rx = Gauge("RX_Packets_of_{}_on_port_{}".format(x,j), "Rx_packets on specific port", registry=registry2)
                Rx.set(z)
                push_to_gateway('localhost:9091', job='RX_PORT', registry=registry2)
            #-------------------------------------
        print("le nombre total de ports est : {}".format(nombre))
except ApiException as e:
    print("Exception when calling FeIDApi->show_fe_id: %s\n" % e)

# Exporter le nombe total des ports et elements
finally:
    nombre_port = Gauge ("total_number_port", " nombre total de ports dans le Clos Network", registry=registry3)
    nombre_port.set(nombre)
    nombre_element = Gauge ("total_number_element", " nombre total d'elements' dans le Clos Network", registry=registry3)
    nombre_element.set(element)
    pushadd_to_gateway('localhost:9091', job='TOTAL_PORT', registry=registry3)
    pushadd_to_gateway('localhost:9091', job='TOTAL_PORT', registry=registry4)
```

Figure 20: Code de recueil de la liste et des activités des ports

Explication :

- Ligne 28-30 : Recueil et projection du nombre d'éléments existants.
- Ligne 33 -35 : Recueil et projection de la liste des éléments identifiés
- Ligne 36-46 : Exportation de la liste des éléments et le nombre de ports de chaque élément vers le PUSHGATEWAY
- Ligne 49-54 : Recueil des nombres de paquets reçus dans chacun des ports existants.
- Ligne 61-67 : Recueil et exportation au PUSHGATEWAY du nombre total d'éléments et de ports.

La figure suivante nous montre le recueil et l'exportation des valeurs du paramètre FOO

Bibliotheque_put > foolist.py

```python
from __future__ import print_function
import swagger_client
import time
import swagger_client
import json
from swagger_client.rest import ApiException
from pprint import pprint
from prometheus_client import Gauge,CollectorRegistry,pushadd_to_gateway
registry = CollectorRegistry()

# Instanciation des Objets
api_instance = swagger_client.FeIDApi()
api_instance1 = swagger_client.FeIDSpecificPortsActivityApi()
api_instance2 = swagger_client.FEIDListOfPortsAndStatsOnSpecificFEApi()
api_instance_foobar = swagger_client.FooValuesApi()

try:
    # Nombre d'elements
    api_response = api_instance.show_fe_id()
    x = len(api_response)
    print("there are {} Fe available".format(x))

    # Exporter les valeurs de foo
    for i in range (0,x):
        FEID = api_response[i].id
        valfoo = api_instance_foobar.foovalue(FEID).foo
        valbar= api_instance_foobar.foovalue(FEID).bar
        print(" FeID {} = {} ".format(i+1, FEID))
        print(" foo of {} = {} ".format(FEID, valfoo))
        valeur_foo = Gauge("FEID_{}".format(FEID), "valeur de foo", registry=registry)
        valeur_foo.set(valfoo)
        pushadd_to_gateway('localhost:9091', job='foo_val', registry=registry)

except ApiException as e:
    print("Exception when calling FeIDApi->show_fe_id: %s\n" % e)
```

Figure 21: Code pour le recueil et l'exportation des valeurs du paramètre FOO

Explication :

- Ligne 1-15 : Importation des bibliothèques et instanciation des objets sus-générés
- Ligne 17-23 : Recueil du nombre d'éléments du réseau
- Ligne 24-29 : Recueil et projection des valeurs de FOO dans chaque élément du réseau
- Ligne 30-32 : exportation vers PUSHGATEWAY des valeurs de FOO de chaque élément

La figure suivante nous montre le recueil et l'exportation des valeurs du paramètre BAR

Bibliotheque_put > barlist.py

```python
from __future__ import print_function
import swagger_client
import time
import swagger_client
import json
from swagger_client.rest import ApiException
from pprint import pprint
from prometheus_client import Gauge,CollectorRegistry,pushadd_to_gateway
registry = CollectorRegistry()

# Instanciaton des objets
api_instance = swagger_client.FeIDApi()
api_instance1 = swagger_client.FeIDSpecificPortsActivityApi()
api_instance2 = swagger_client.FEIDListOfPortsAndStatsOnSpecificFEApi()
api_instance_foobar = swagger_client.FooValuesApi()

try:
    # Nombre d'elements
    api_response = api_instance.show_fe_id()
    x = len(api_response)
    print("there are {} Fe available".format(x))

    # Exportateur des valeurs de bar de chaque element
    for i in range (0,x):
        FEID = api_response[i].id
        valfoo = api_instance_foobar.foovalue(FEID).foo
        valbar= api_instance_foobar.foovalue(FEID).bar
        print(" FeID {} = {} ".format(i+1, FEID))
        print(" bar of {} = {} ".format(FEID, valbar))
        valeur_bar = Gauge("FEID_{}".format(FEID), "valeur de bar", registry=registry)
        valeur_bar.set(valbar)
        pushadd_to_gateway('localhost:9091', job='bar_val', registry=registry)

except ApiException as e:
    print("Exception when calling FeIDApi->show_fe_id: %s\n" % e)
```

Figure 22: Code pour le recueil et exportation des valeurs du paramètre BAR

Explication :

- Ligne 1-15 : Importation des bibliothèques et instanciation des objets sus-générés
- Ligne 17-23 : Recueil du nombre d'éléments du réseau
- Ligne 24-29 : Recueil et projection des valeurs de bar dans chaque élément du réseau
- Ligne 30-32 : exportation vers PUSHGATEWAY des valeurs de BAR de chaque élément.

6.3.1.2 Développer l'exportateur, pour configuration des valeurs des paramètres de configuration du réseau.

Notons que, le développement de cette application de configuration des FOO/BAR constitue un modèle de développement typique pour toutes autres configurations du réseau à savoir les configurations des paramètres comme les : maximum transmission unit (mtu), statut des ports (port fonctionnel ou non), priorité des trafics, etc.

La figure suivante montre un exportateur de configuration de FOO/BAR. Le développement se fait de façon Orientée Objet, en accédant la bibliothèque d'objets SWAGGER_CLIENT.

Cette application nous permet de :

- Recueillir l'actuelle valeur des FOO/BAR d'un port désigné
- Configurer les nouvelles valeurs de FOO/BAR
- Exporter ces valeurs configurées au PUSHGATEWAY

La figure suivante nous montre les codes sources pour ces configurations.

Bibliotheque_put > foobarconfig.py

```python
from __future__ import print_function
import swagger_client
import time
import swagger_client
import json
from swagger_client.rest import ApiException
from pprint import pprint
from prometheus_client import Gauge,CollectorRegistry,pushadd_to_gateway
registry = CollectorRegistry()

# Instanciation des objets
api_instance = swagger_client.FeIDApi()
api_instance1 = swagger_client.FeIDSpecificPortsActivityApi()
api_instance2 = swagger_client.FEIDListOfPortsAndStatsOnSpecificFEApi()
api_instance_foobar = swagger_client.FooValuesApi()
api_instance_bar = swagger_client.BarValuesApi()

```

```
try:
    # Nombre d'elements
    api_response = api_instance.show_fe_id()
    x = len(api_response)
    print("there are {} Fe available".format(x))
    for i in range (0,x):
        print(" FeID {} = {} ".format(i+1, api_response[i].id))

    # Configuration des valeurs de foo et bar

    test = input("entrer le numero de FE : ")
    valfoo = api_instance_foobar.foovalue(test).foo
    valbar= api_instance_foobar.foovalue(test).bar

    # Configuration de foo
    print(" La valeur actuelle de foo est : {}".format(valfoo))
    choix = input(" voulez vous changez la valeur de foo ? si OUI: tapez 1 / si NON: tapez 0 :")
    if choix == 1:
        new = input("Veuillez entrer la nouvelle valeur: ")
        newvalfoo= str(new)
        api_instance_foobar.updatefoo(newvalfoo, test)
        print(" La nouvelle Valeur de foo du FEId {} est : {}".format(test,api_instance_foobar.foovalue(test).foo))
    else :
        print (" D'accord, la valeur reste le meme")

    # Configuration de bar
    print(" La valeur actuelle de bar est : {}".format(valbar))
    choix1 = input(" voulez vous changez la valeur de bar ? si OUI: tapez 1 / si NON: tapez 0 :")
    if choix1 == 1:
        new = input("Veuillez entrer la nouvelle valeur: ")
        newvalbar= str(new)
        api_instance_bar.updatebar(newvalbar, test)
        print(" La nouvelle Valeur de bar du FEId {} est : {}".format(test,api_instance_foobar.foovalue(test).bar))

    else :
        print (" D'accord, la valeur reste le meme")
    newfoo = api_instance_foobar.foovalue(test).foo
    newbar = api_instance_foobar.foovalue(test).bar

except ApiException as e:
    print("Exception when calling FeIDApi->show_fe_id: %s\n" % e)

# Exportation des valeurs de foof et bar
finally:
    valeur_foo = Gauge("FE_foo_of_{}".format(test), "nouvelle valeur de foo", registry=registry)
    valeur_foo.set(newfoo)
    valeur_bar = Gauge("FE_bar_of_{}".format(test), "nouvelle valeur de bar", registry=registry)
    valeur_bar.set(newbar)
    pushadd_to_gateway('localhost:9091', job='FEID_foobar_val', registry=registry)
```

Figure 23: Code source configurations de FOO/BAR

Explication :

- Ligne 1-16 : Importation des bibliothèques et instanciation des objets sus-générés
- Ligne 18-26 : Recueil et projection des éléments du réseau
- Ligne 26-41 : Recueil et configuration des valeurs de FOO dans l'élément désigné
- Ligne 43-50 : Recueil et configuration des valeurs de BAR dans l'élément désigné
- Ligne 52- 66 : exportation vers PUSHGATEWAY des nouvelles valeurs de FOO/BAR de l'élément désigné.

Remarque :

Toutes les tâches de récupération se font d'une manière très courte, et dans notre travail, les recueils se font toutes les 2 secondes [24].

La figure suivante nous montre les commandes permettant la périodicité des recueils de données.

```
while true
do
        /usr/bin/python  /home/yvon/Documents/memoire/Bibliotheque_get/listFEId.py & /usr/bin/python  /home/yvon/Documents/memoire/
Bibliotheque_put/foolist.py & /usr/bin/python  /home/yvon/Documents/memoire/Bibliotheque_put/barlist.py

        sleep 2
done
```

Figure 24: Batch pour une récupération de données périodique

6.3.2 Prendre en main les fonctionnalités de PROMETHEUS

- ✓ Implémentation et configuration PROMETHEUS Version 2.12.0.
- ✓ Maîtriser les récupérations (*PULL*) des différentes métriques exposées par PUSHGATEWAY des éléments du réseau (FE- *Forwarding Elements*) et leur stockage par le TSDB (*Time Séries Data Base*) de PROMETHEUS.
- ✓ Connaitre le mécanisme du PUSHGATEWAY, qui constitue une passerelle pour stocker les données métriques provenant des tâches à très courtes durées.

Remarque :
Dans ce projet, ce sont les données exportées par nos exportateurs qui sont stockées temporairement dans PUSHGATEWAY avant d'être récupérées par PROMETHEUS.

La figure suivante nous montre la liste des tâches exportées vers PUSHGATEWAY

Figure 25: Listes des tâches exportées vers PUSHGATEWAY

Explication : Tableau expliquant les différents « job= "….." »

Job (tâche)	Description
foo_val	Liste des valeurs de FOO dans chaque élément du réseau
Duration	Durée des recueils + exportations des ports et des paquets reçus.
NUMBER_PORT	Nombre de ports existants dans chaque élément du réseau
TOTAL_PORT	Nombres totaux des ports et des éléments existants
FEID_LIST	Liste des éléments ainsi que leurs identités
RX_PORT	Liste des vitesses de réception des données dans chaque port
Bar_val	Liste des valeurs de bar dans chaque élément du réseau

Les figures suivantes nous montrent des détails des tâches spécifiques.

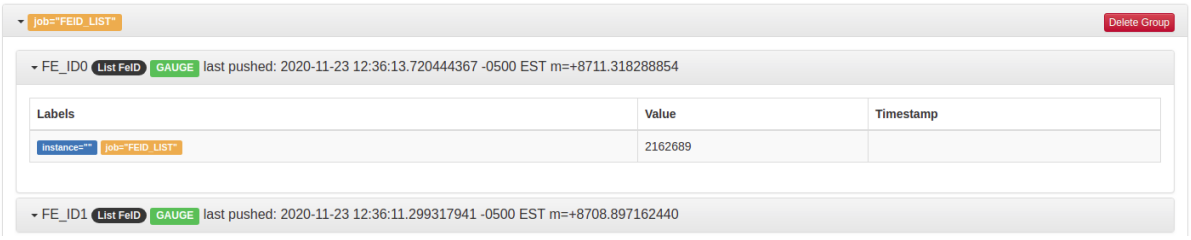

Figure 26: Détails partiels d'une tâche spécifique « FEID_LIST »

Explication :

- Dans la tâche FEID_LIST, on distingue que FE_IDO est le premier élément recueilli et qui a pour identité ID = 2162689

Figure 27: Détails partiels d'une tâche spécifique «RX_PORT »

Explication :

- Dans la tâche RX_PORT, on distingue RX_Packets_of_2162689_on _port_0 qui définit le nombre de paquets reçus dans le premier port ou port 0 dans l'élément ID = 2162689 est : 14976, le 23/11/20 à 12:36.

La figure suivante nous montre la récupération (PULL) des données par PROMETHEUS dans le PUSHGATEWAY.

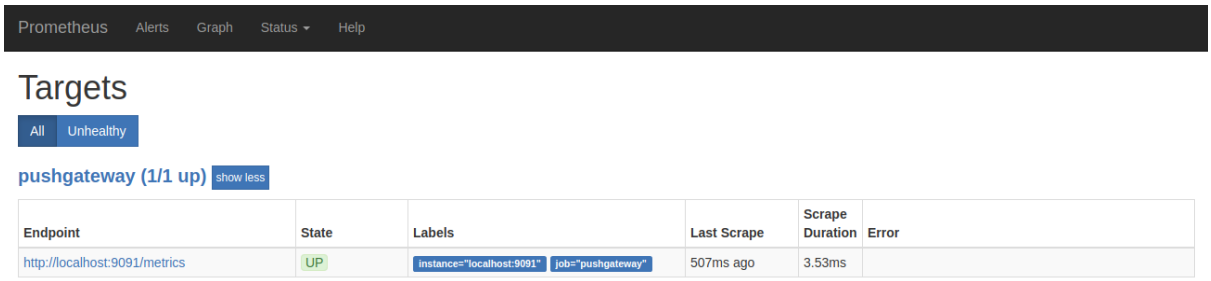

Figure 28: Serveur PROMETHEUS ayant comme cible PUSHGATEWAY

6.3.3 Analyse des données par GRAFANA

✓ Implémentation et configuration de GRAFANA.
✓ Familiarisation avec les tableaux de bord, et les outils disponibles sur GRAFANA.
✓ Analyse des données dans PROMETHEUS avec GRAFANA

Notons que dans cette étape, les requêtes pour accéder aux données dans PROMETHEUS sont écrites par des langages qui lui sont propres, c'est le « PromQl–*Prometheus Query Langage* ».

La figure suivante nous montre l'analyse des données de notre réseau avec GRAFANA.

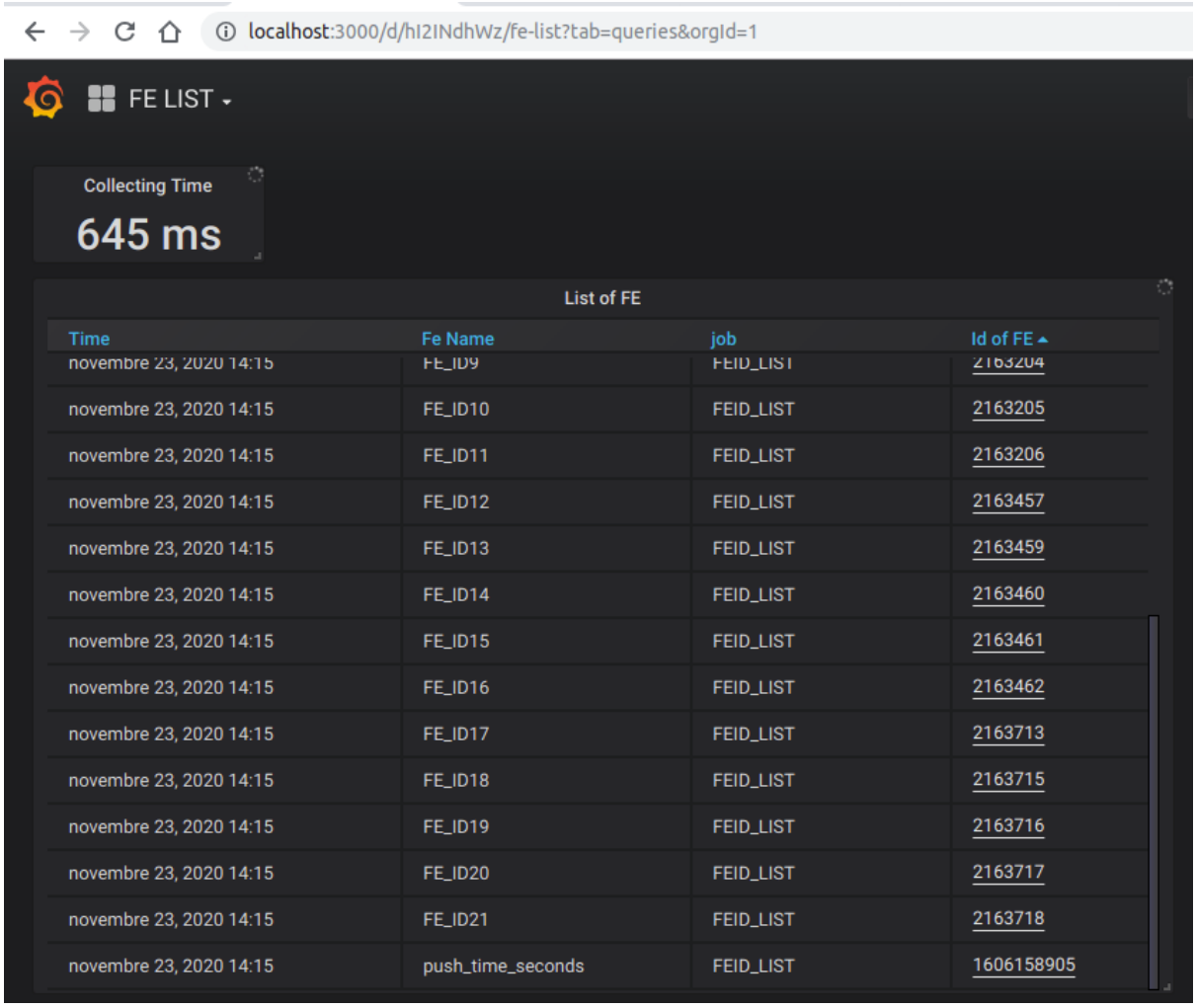

Figure 29: Visualisation graphique des tâches « FEID_list » et « Duration »

Explication :

- Collecting time : Durée de recueil [25] des listes des ports et des paquets reçus qui est à 645 ms. Notons que ce temps de recueil dépend de l'étendu du réseau clos.
- List of FE : Liste des éléments existants ainsi que leurs identités respectives.
- Requête PromQl : Yvonjob_duration_seconds{job="duration"}/{job="FEID_LIST"}

La figure suivante nous montre les nombres totaux des ports, des éléments ainsi que le temps de collecte sus-cité ainsi que le nombre de port existant dans chaque élément.

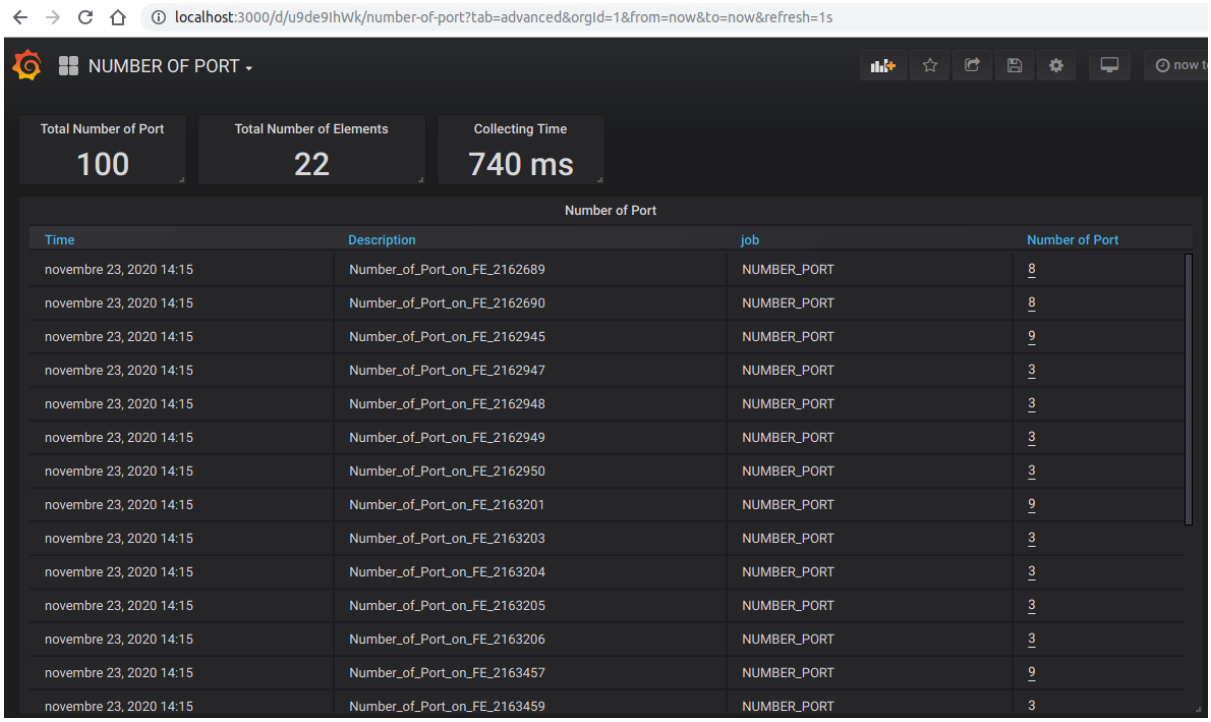

Figure 30: Visualisation graphique des tâches « NUMBER_PORT, TOTAL_PORT, Duration»

Explication :

- Total Number of Port : le nombre total des ports existants dans tout le réseau est 100
- Total Number of Elements : le nombre total des éléments du réseau est 22
- Number of port : le nombre de port pour chaque élément, on peut distinguer celui de l'élément 2162689 qui est égal à 8 et celui de l'élément 2162945 qui est égal à 9.
- Requêtes PromQl :
    - total_number_port{job="TOTAL_PORT"}
    - total_numer_elementjob="TOTAL_PORT"}
    - Yvonjob_duration_seconds{job="duration"}
    - {job="NUMBER_PORT"}

La figure suivante nous montre une interprétation graphique du nombre de paquets reçus en moyenne par minute de quelques ports spécifiques.

Figure 31: Interprétation graphique de la tâche « RX_PORT »

Explication :

- Spine 2 Port Activity : Le nombre de paquets reçu en moyenne/minute des 4 ports du deuxième commutateur. En pratique, ce sont les ports qui relient le commutateur aux 4 racks.
- Rx Port 3 of each Feid : Le nombre de paquets reçu en moyenne/minute de chaque port numéro 3 des 4 racks.
- Spine 1 Port Activity : Le nombre de paquets reçu en moyenne/minute des 4 ports du premier commutateur. En pratique, ce sont les ports qui relient le commutateur aux 4 racks.
- Les axes horizontaux des graphes représentent les axes de temps
- Les axes verticaux représentent les moyennes/minute, des nombres de paquets reçus.
- Requêtes PromQl typique:
    - rate(RX_Packets_of_2162698_on_port_4{job="RX_PORT"}[1m])
    - rate(RX_Packets_of_2162945_on_port_3{job="RX_PORT"}[1m])
    - rate(RX_Packets_of_2162689_on_port_5{job="RX_PORT"}[1m])

La figure suivante projette les valeurs de FOO/BAR de chaque élément du réseau.

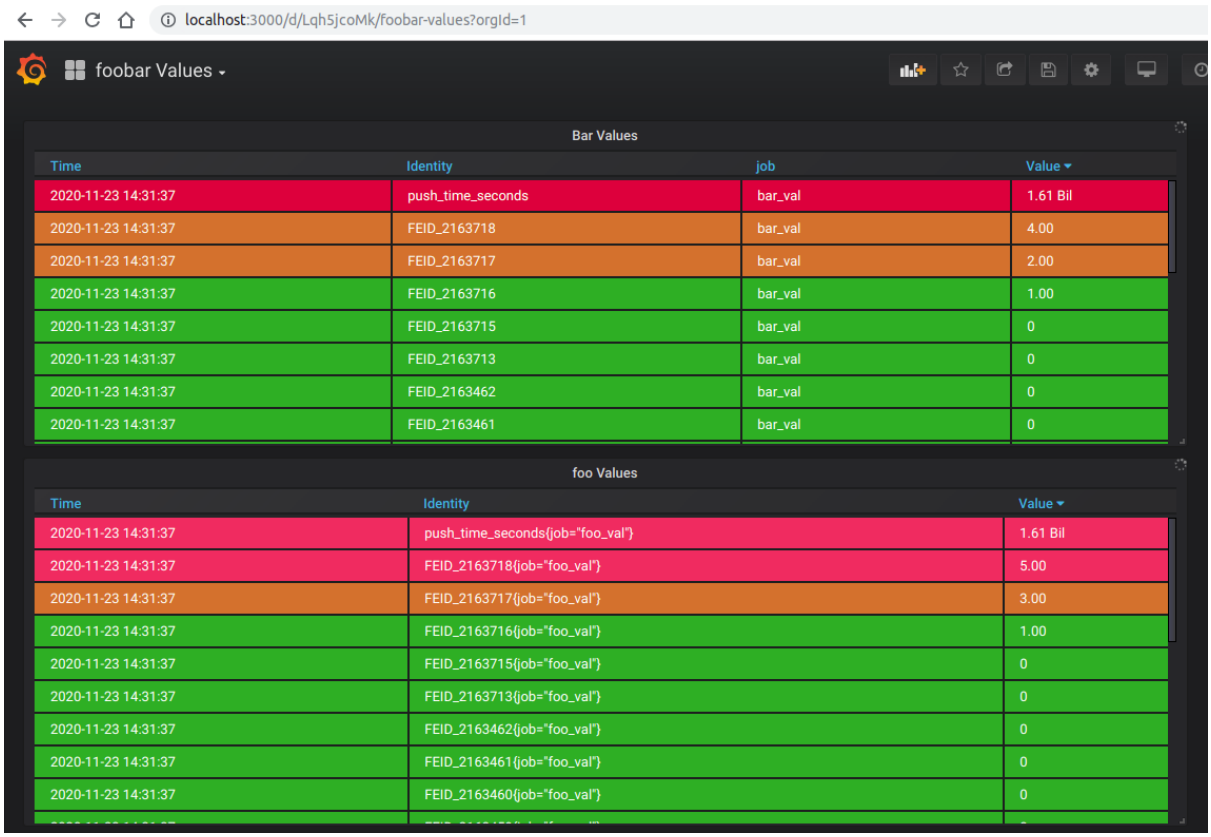

Figure 32: Visualisation des valeurs de FOO/BAR

Explication :

- Bar Values : Projection des valeurs de Bar de chaque élément du réseau. On peut distinguer qu'à 14 :31 le 23/11/20, la valeur de Bar de l'élément 2163718 est à 4 et celle de 2163717 est à 2.
- Foo Values : Projection des valeurs de Foo de chaque élément du réseau. On peut distinguer qu'à 14 :31 le 23/11/20, la valeur de Bar de l'élément 2163718 est à 5 et celle de 2163717 est à 3
- Requêtes PromQl :
    - {job= "bar_val"}
    - {job="fo_val"}
- Les codes couleurs sont définis comme tels :
    - Vert : 0<x<2
    - Orange : 2<x<=4
    - Rouge : 4<x

6.4 Récapitulation

A titre d'évaluation de l'OAS, les points suivants ont été vérifiés :

- Avec OAS 3.0, les spécifications des services sont vraiment explicites.
- La bibliothèque d'objets générée nous permet, non seulement, d'accéder aux services de façon orienté objet, mais en plus, nous met à disposition des exemples d'utilisation pour les objets existants.
- Le développement des exportateurs, qui est une partie clé de notre couche d'application, a été facilité grâce à la bibliothèque.
- OAS est efficace dans le développement des applications autant les récupérations (GET) des données que les configurations (PUT/SET//POST) des paramètres.

Bref, nous avons pu bénéficier des avantages de l'OAS dans notre SDN.

La figure suivante nous permet de récapituler les travaux faits, et nous avons mis en évidence les développements en couleur jaune.

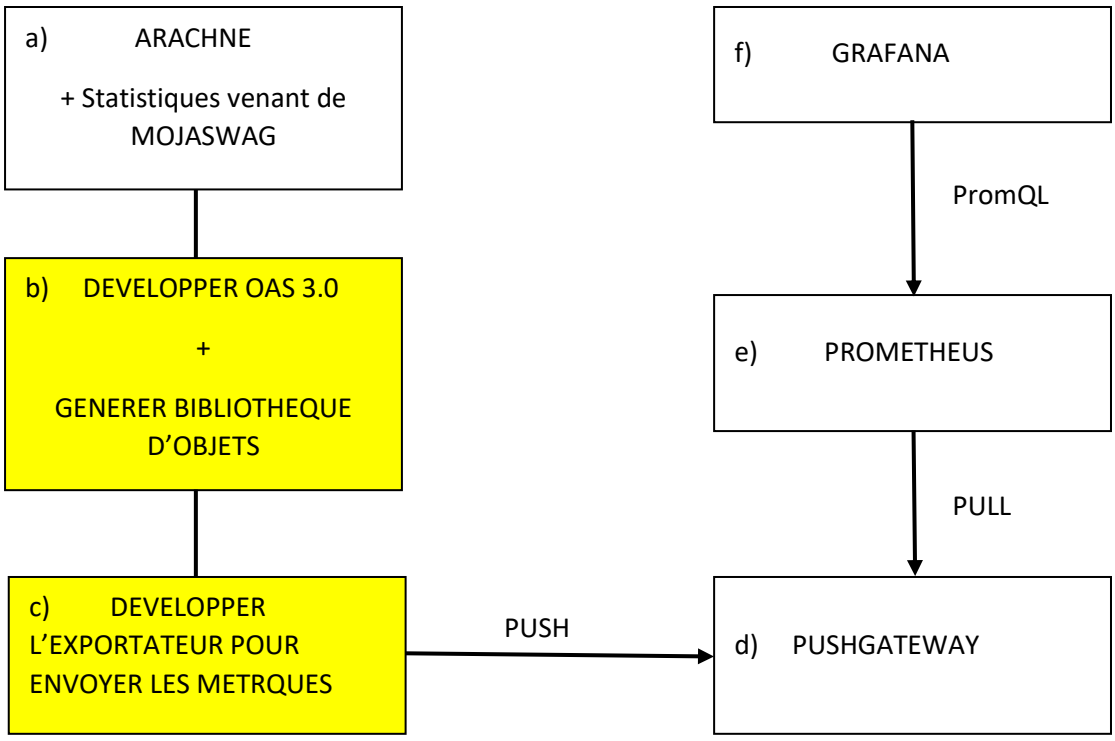

Figure 33: Flux du travail pour tester l'insertion d'OAS

a) Simulation du réseau Clos avec ARACHNE.
b) Accès à l'interface Nord MOJASWAG, découverte et compréhension des services.
c) Développement d'OAS, puis générer la bibliothèque d'objets.
d) Développement des exportateurs de données vers PUSHGATEWAY
e) Récupération des données par le serveur PROMETHEUS
f) Visualisation/Analyse graphique des données stockées dans PROMETHEUS

Chapitre 7

Conclusion

Dans le contexte SDN, un des défis majeurs pour défendre le terme « programmable » est d'avoir une interface Nord normalisée qui permet aux acquéreurs d'une telle technologie d'agir à leurs guise et selon leurs besoins dans l'amélioration ou la modification des fonctionnalités de leurs applications de gestion du réseau.

En se référant à notre recherche et à nos résultats: on réalise qu'une nouvelle couche d'OAS, intégrée dans une application SDN, permet d'avoir un ensemble de système ouvert, flexible, et surtout gratuit (accessible au public). En plus (comme bonus), nous avons développé des applications exportatrices des données métriques, de façon Orientée Objet.

En conclusion nous pouvons affirmer que OAS est efficace, pas seulement dans le développement des applications de gestion de base de données mais aussi dans le domaine du développement du système de gestion de réseaux informatiques.

Notre approche, avec des investissements supplémentaires pourrait être un candidat potentiel pour une standardisation de l'interface Nord du SDN.

Néanmoins, comme nous l'avons évoqué dans ce travail, les concepteurs de SDN devront s'aligner sur un même principe pour rendre le SDN programmable autant du coté client que du côté concepteur.

L'efficacité d'OAS avec un SDN, dont l'interface SUD est ForCES, est concluante mais serait-il envisageable d'en faire autant avec les SDN basés sur d'autres protocoles?

D'après nous, autant que les interfaces Nord sont présentées avec une base API REST [26] ce qui est généralement le cas [27], OAS serait adéquat.

Donc, dans le cadre des perspectives futures, valider cette approche avec les SDN basés sur d'autres protocoles comme OPENFLOW qui est actuellement le protocole SDN le plus populaire [27], vaudrait plus d'arguments pour un projet de standardisation de l'interface Nord.

Références

[1] M. Z. Abdullah, N. A. Al-awaad, F. W. Hussein (2018), "Performance Evaluation and Comparison of Software Defined Networks Controllers", International Journal of Scientific Engineering and Science, Vol 2. Issue 11, pp 45-50, 2018

[2] D. Kreutz, Fernando M.V. Ramos, P. Verissimo (2014), C. Rothenberg, S. Azodolmolky, S. Uhlig, " SDN : A comprehensive survey", Proceedings of the Institute of Electrical and Electronics Engineers (IEEE), vol: 103, p: [18-20], décembre 2014.

[3] V. Tijare, D. VaSudevan (2016), "The northbund APIs of SDN", International journal of engineering sciences & research technology, DOI: 10.5281/zenodo.160891, Octobre 2016.

[4] E. Haleplidis, Ed., Jamal Hadi Salim (2015), "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, International Research Task Force, ISSN: 2070-1721, Janvier 2015.

[5] M. Boucadair, C. Jacquenet (2014), "SDN: A perspective from within a service provider environnement", RFC 7149, International Engineering Task Force, ISSN: 2070-1721, Mars 2014.

[6] Y. A. Matnee, C. H. Abooddy, Z. Q. Mohammed (2018), "Analyzing Methods and Opportunities in Software Defined (SDN) Networks for Data Traffic Optimizations", International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 6 Issue: 1, Janvier 2018.

[7] N. Feamster, J. Rexford, E. Zegura (2014), "The road to SDN: an intellectual history of programmable networks." SIGCOMM Comput. Commun. Rev. 44, 2 (April 2014), 87–98.

[8] M. Meng, S. Steinhardt, A. Shubert (2017), "Application Programming Interface Documentation ", Journal of technical writing and communication, Vol.48(3), P [295-330].

[9] Open Network Foundation (2015), "Openflow switch specification", Protocol version 0x06, 26 Mars 2015.

URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[10] J. Halpern, J. Hadi Salim (2010), “Forwarding and Control Element Separation (ForCES), Forwarding Element model”, RFC 5812, Internet Engineering Task Force, ISSN: 2070—1721, Mars 2010.

[11] M. Aouadj (2016), “Airnet : Le modèle de virtualisation comme plan de contrôle pour les réseaux programmables”, Thèse en vue de l’obtention d’un doctorat, Université de Toulouse, p.34, Novembre 2016.

[12] F. Pakzad, M. Portmann, W. Lum Tan, J. Indulska (2016), “Efficient topology discovery in OpenFlow-based Software Defined Networks”, Computer Communications,Volume 77, Pages 52-61, ISSN 0140-3664.

[13] SMARTBEAR (2019), “Open API Specification”, Definition specification appendix A: revision History, version 3.0.3, consulté en ligne en juin 2019.
URL: https://swagger.io/specification/

[14] E. Mc Kean (2018), “Evrything you need to know about Open API 3.0”, Lead Dev New York 2018, consulté en Octobre 2019.
URL: https://youtu.be/NF15GTBEb0K

[15] O. Ben-Kiki C. Evans (2009), “YAML ain’t markup language”, version 1.2, 3ème édition, octobre 2009, consulté en Mars 2019.
URL: https://yaml.org/spec/1.2/spec.html,

[16] D. Crockford (2017), “Introducing JSON”, ECMA-404 The JSON Data Interchange Standard, consulté en ligne en Mai 2019.
URL: https://www.json.org/json-en.html,

[17] R. Ratovsky (2018), “webinard”, smartbear community, consulté en Janvier 2020.
URL: https://youtu.be/KcQJ3Q7h_Fo

[18] K. Vasudevan, R. Pinkham (2017), “ OpenAPI 3.0: How to Design and Document APIs with the Latest Open API Specification 3.0”, smartbear septembre 2017, vu en Aout 2020.
URL: https://youtu.be/6kwmW_p_Tig

[19] J. Hadi Salim et A. Aring (2017), “Arachne: Large Scale Data Center SDN Testing”, Netdev conf 2.2, The Technical Conference on linux Networking, Nov 2017.

[20] MOJATATU Network Company, Owner: Jamal Hadi Salim, Ottawa, Ontario, Canada.
URL: http://www.mojatatu.info.

[21] Prometheus Authors (2014-2020) Documentation distribué sous CC-BY-4.0, consulté en Janvier –Juin 2019.
URL: https://prometheus.io/docs/

[22] Grafana labs, “GRAFANA”, consulté en ligne en mars-juin 2019.
URL: http://grafana.com

[23] L. Drake, Python Software Fondation, “Python”, consulté en ligne en Aout 2019.
URL: https://python.org

[24] Expert éminent sénior (2014), “ Script permettant de lancer une commande toutes les X secondes “, forum du 04 Avril 2014 16 :47, consulté en Mars 2020
URL: https://www.developpez.net/forums/d1413443/systemes/linux/shell-commandes-gnu/script-permettant-lancer-commande-toutes-x-secondes/

[25] Brian Brazil (2015), “Monitoring Batch Jobs in Python”, A blog on monitoring, scale and operational Sanity, consulté en février 2020.
URL: https://www.robustperception.io/monitoring-batch-jobs-in-python

[26] Zell liew (2018), “Understanding and Using REST APIs”, Smashing magazine, 17 janvier 2018, consulté en septembre 2020
URL: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/

[27] L. Mamushiane, A. Lysko and S. Dlamini (2018), "A comparative evaluation of the performance of popular SDN controllers," 2018 Wireless Days (WD), Dubai, 2018, pp. 54-59, doi: 10.1109/WD.2018.8361694.

[28] A.Arnaud, A. Amelina, A.P. AINA (2014), “Virtualisation & Partage des charges”, AFNOG 2014, Track SS-F : Services Internet Evolutifs, 2014.

[29] Digital guide IONOS (2019), “Les conteneurs informatiques : la nouvelle virtualisation”, consulté en Novembre 2019.
URL: https://www.ionos.fr/digitalguide/serveur/know-how/conteneurs-informatiques-virtualisation-sans-emulation/

[30] SoundCloud Entreprise (2007), Fondateur: Fross, Berlin, Allemagne, Consulté en Octobre 2019.
URL: https://soundcloud.com/

[31] A. Sabathier (2018), “Go : le langage Informatique par Google’, ELP-2018 Ecosystème des langages de programmation, consulté en Mai 2019.
URL: https://medium.com/elp-2018/go-le-langage-informatique-par-google-8d574f6fd782

Annexe A - ARACHNE

ARACHNE est une application de simulation d'un réseau clos qui est développé par la société MOJATATU. C'est l'outil dont nous utilisons pour simuler nos couches de données et de contrôles.

En se référant à l'architecture du SDN, ARACHNE simule la couche infrastructure et la couche contrôle. Ce programme est ouvert et gratuit. Ses caractéristiques et ses capacités sont les suivants : implémenter sur un ordinateur basé avec un OS linux, simuler un réseau fermé, et générer des trafics internes.

Pour tester l'échelle et la robustesse d'un grand déploiement SDN, il faut un gros investissement dans des nombreux équipements comme les commutateurs, et les serveurs réels. Une alternative est de basculer avec la configuration basée sur les Machines Virtuelles - VM [28]. Les machines virtuelles, dans une simulation d'un réseau clos de très large envergure, sont encore très couteuses vu qu'on aura besoin de plusieurs machines physiques pour mettre à disposition des milliers voire des millions d'éléments du réseau, ce qui amène MOJATATU à se tourner vers la configuration des conteneurs [29] qui réduit énormément le coût et la quantité d'équipements à déployer.

1- Déploiement d'ARACHNE - Infrastructure réseau fermé (Clos Network)

Les environnements SDN sont généralement déployés sur des architectures de centres de données; et actuellement, le déploiement le plus populaire des centres de données repose sur les réseaux fermés. Pour cette raison, MOJATATU a choisi d'utiliser cette architecture en tant qu'infrastructure permettant d'atteindre leurs objectifs.

Les réseaux fermés et SDN sont le résultat de générations de travail dans le monde de la téléphonie, dont ci-dessous l'historique :

Dans les années 50, les réseaux fermés étaient une influence du monde de la technologie, qui connectaient les systèmes téléphoniques: si nous imaginons un hôte comme un téléphone et les commutateurs Ethernet comme des commutateurs vocaux, nous pouvons visualiser comment les appels téléphoniques ont été basculés de l'appelant à l'appelé.

Ensuite, dans les années 70, le monde du téléphone a également beaucoup influencé le SDN. Les réseaux téléphoniques ont introduit le concept de séparation de la signalisation de contrôle et de la voix de transmission de données en deux réseaux distincts.

Dans le contexte d'ARACHNE, on utilise des commutateurs et des hôtes (dépendants du déploiement) comme nœuds de transfert de données. Les équipements des centres de données rencontrés, jusqu'à présent (commutateurs ou hôtes), sont équipés d'un ou de plusieurs ports de gestion. Ces ports sont utilisés pour déployer l'infrastructure de contrôle.

Avec ARACHNE, le réseau fermé fournit une configuration de commutation à plusieurs étages qui sont intéressante pour deux raisons : son utilisation dans les centres de données implique la similitude de ce déploiement dans le monde réel, d'une part. Et la possibilité de faire passer l'infrastructure d'une petite construction à une construction extrêmement volumineuse, d'autre part.

Au niveau de sa topologie, plusieurs nœuds peuvent être ajoutés sans interrompre une configuration existante. Cela signifie la possibilité de tester progressivement, différents niveaux d'échelle, sans annuler une configuration de test en cours.

Les réseaux fermés peuvent être généralisés à un nombre impair d'étages de commutation ; En général, 3 et 5 étapes sont courantes.

1.1 Réseau Fermé à 3 étapes

Une encapsulation de réseau fermé à 3 étapes de base est appelée PdD/PoD (Point de Déploiement/ Point of Déploiement). A noter que dans un PdD,

- Un rack d'hôtes est connecté à un commutateur feuille, appelé commutateur Top of Rack (TOR).
- Chaque feuille est connectée à chaque épine de cette PdD. Le trafic provenant d'un hôte dans un rack doit franchir trois sauts / étapes de commutateur (d'où la convention à 3 étapes) pour atteindre un hôte sur un autre rack au sein de la PdD:
    - ✓ Feuille locale à la colonne vertébrale
    - ✓ Épine à la feuille à distance
    - ✓ Feuille distante à l'hôte distant

La Figure suivante montre une configuration PdD très basique avec 4 racks, chaque rack transporte une douzaine d'hôtes et un commutateur feuille.

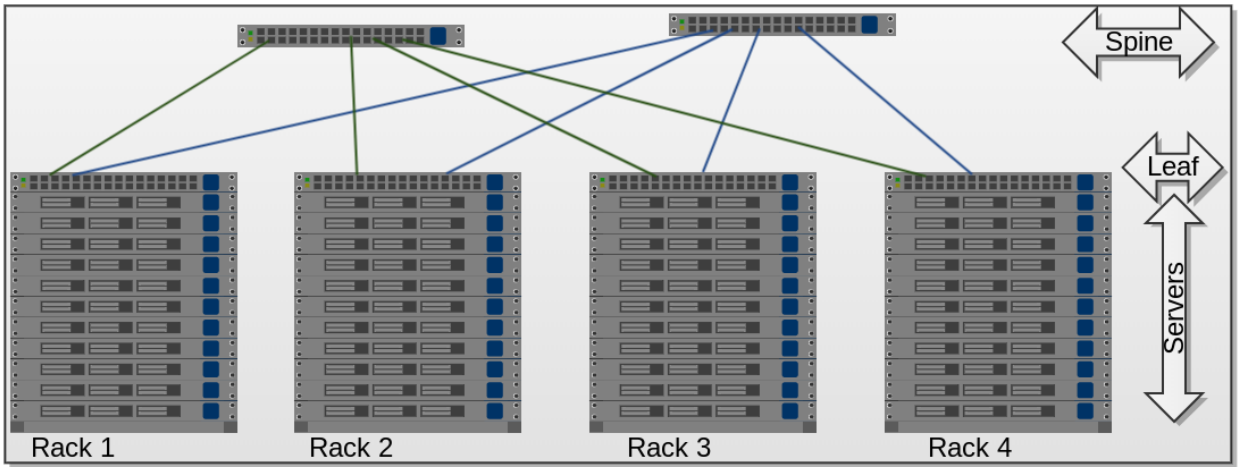

Figure 34: Réseau fermé à 3 étapes

Du point de vue de l'échelle / de la modularité et de l'automatisation, il est possible d'imaginer un nouveau rack d'hôtes préconfiguré avec son commutateur feuille; qu'on peut directement le connecter au commutateur feuille active, qui est toujours connectées à toutes les épines. Donc, le nouveau rack peut être utilisé immédiatement, sans réinitialiser le système.

1.2- Réseau fermé à 5 étapes ou Zone

Un réseau fermé peut également être mis à l'échelle en interconnectant des PdD; cela nécessiterait une couche de commutateurs au-dessus des épines pour connecter les PdD les uns avec les autres. Une telle configuration est appelée une zone. Il convient de noter que les commutateurs inter-PdD ont beaucoup d'autres conventions de dénomination dans l'industrie, telles que, structure, super-colonne vertébrale, noyau, etc. Dans cette convention de dénomination, les commutateurs sont appelés inter-PdD commutateurs de zone. Dans un réseau fermé à 5 étapes, les paquets émanant d'un hôte d'une PdD doivent prendre 5 sauts pour atteindre un hôte d'une PdD différente.

Une configuration typique de la manière dont les zones sont connectées via les commutateurs de zone est similaire aux connexions feuille à colonne vertébrale, c'est-à-dire, que chaque commutateur de colonne vertébrale de chaque PdD se connecte à tous les commutateurs de zone. Une telle configuration est illustrée par la Figure 35.

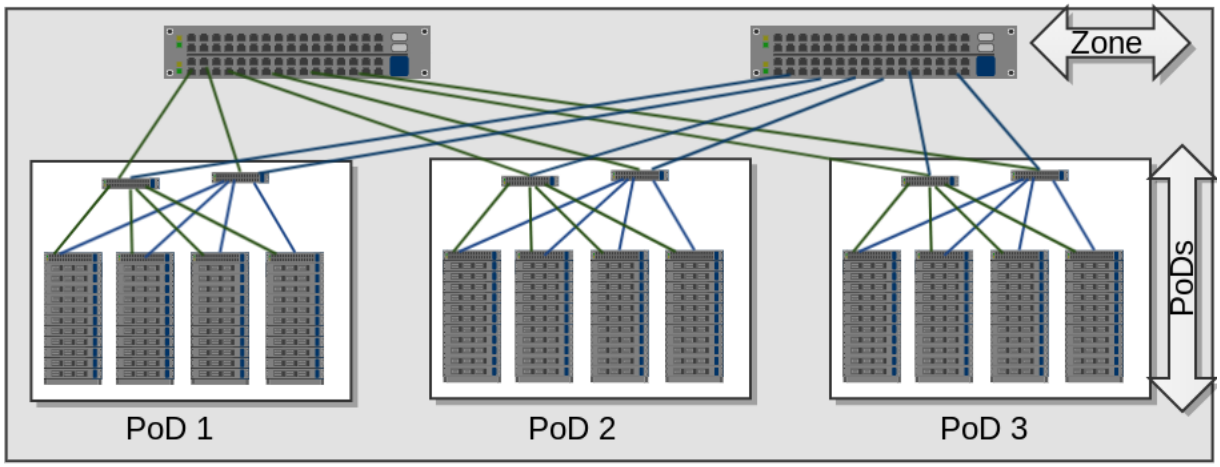

Figure 35: Réseau fermé à 5 étapes

Du point de vue de la montée en charge / de la modularité et de l'automatisation, on peut imaginer prendre un tout nouveau PdD préconfiguré de l'usine et connecter ses commutateurs avec les commutateurs de zone, confirmer, et le nouveau élément est actif.

2- **Communication avec ARACHNE.**

Comme évoqué un peu plus haut, le protocole de communication entre la couche contrôle et ARACHNE est ForCES.

***3*- Lancement d'ARACHNE avec le plugin ForCES**

- Pré-configuration environnement avec la commande : Sudo chmod 666 dev/kvm/
- Commande : ~/arachne$ ./arachne -p ./arachne_plugins_forces/

4- MOJASWAG

MOJASWAG est un serveur développé par la compagnie MOJATATU qui met à disposition les services permettant à ARACHNE d'exposer les statistiques d'activités des ports et leurs caractéristiques, recensées dans le réseau fermé, comme la vitesse de transmission et de réception, ainsi que les identifications de chacun des ports existants.
Dans la pratique SDN, c'est le serveur d'Interface Nord dont nous utilisons pour investiguer les services disponibles.

5- Lancement de MOJASWAG

- Commande: ~/arachne/SDK/ForCES-SDK-build_all_x86_64_trusty-2477$ mojaswag -p 25.0.0.1

Annexe B - PROMETHEUS

PROMETHEUS est un outil ouvert de surveillance et de surveillance des systèmes, créé à l'origine par SoundCloud [30]. Depuis sa création en 2012, de nombreuses sociétés et organisations ont adopté PROMETHEUS. Le projet compte une communauté de développeurs, et d'utilisateurs très actives. Aujourd'hui, ce projet open source autonome et maintenu indépendamment de toute entreprise est disponible.
Dans notre cas, c'est le serveur dont nous utilisons pour récupérer et stocker les métriques des éléments d'ARACHNE.

1- Caractéristiques

Nous énumérons ci-après les principales caractéristiques de PROMETHEUS:

- Il est un modèle de données multidimensionnelles, avec des données de série chronologique, identifiées par un nom de métrique, et des paires clé / valeur.
- Il met à disposition un langage de requête flexible, appelé PromQL, pour exploiter cette dimensionnalité.
- Il n'a pas de dépendance sur le stockage distribué; les nœuds de serveur uniques sont autonomes.
- Il assure la collecte des séries chronologiques via un modèle d'extraction de type http.
- Il permet le renvoi des séries chronologiques, via une passerelle intermédiaire qui s'appelle PUSHGATEWAY.
- Il possède plusieurs supports graphiques et de tableaux de bord.

2- Composants

L'architecture de PROMETHEUS est constituée de plusieurs composants, dont beaucoup sont facultatifs:

- Le serveur principal PROMETHEUS qui récupère et stocke les données de séries chronologiques.
- La bibliothèque client pour instrumenter le code d'application.
- Une passerelle push (PUSHGATEWAY) pour soutenir les tâches de courte durée.

- Les exportateurs spéciaux de métriques, tels que HAProxy, StatsD, Graphite, etc.
- Un gestionnaire d'alerte pour gérer les alertes.
- Les divers outils de support.

La plupart des composants PROMETHEUS sont écrits en langage Go [31], ce qui facilite leur construction et leur déploiement en tant que fichiers binaires statiques.

3- Architecture

La figure suivante illustre l'architecture de PROMETHEUS et certains de ses composants voisins:

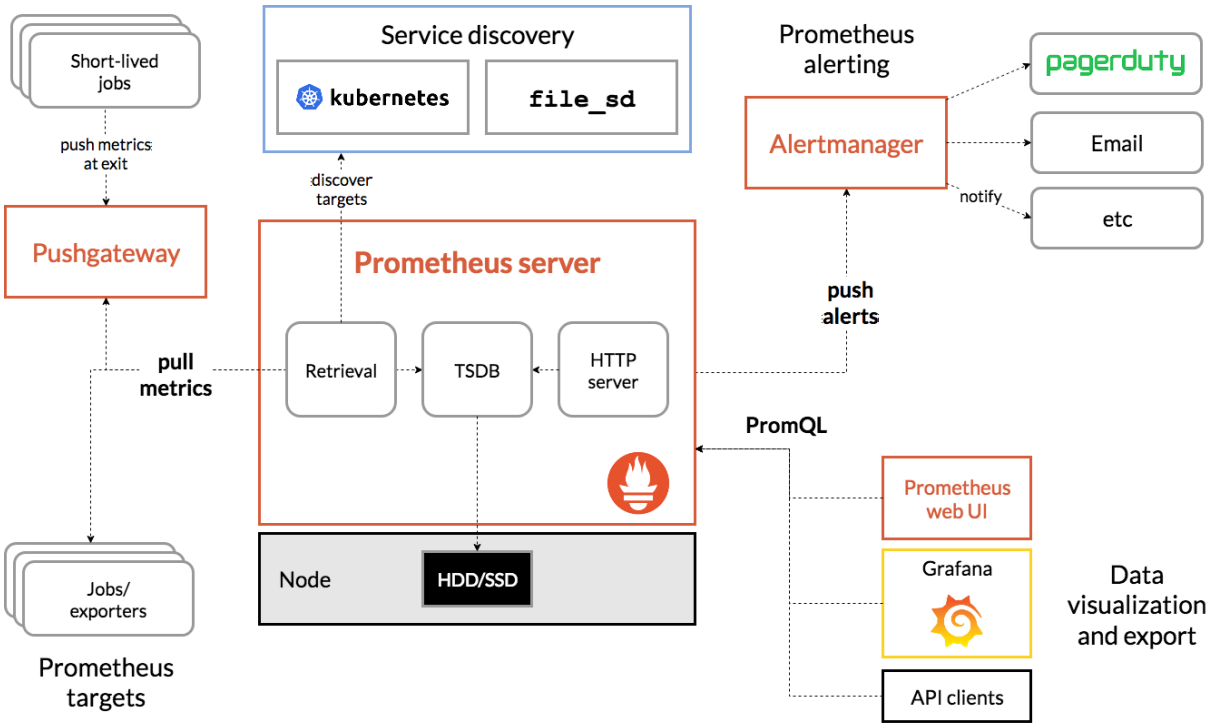

Figure 36: Architecture de PROMETHEUS

PROMETHEUS collecte les métriques des activités applicatives et/ou matérielles, directement ou via une passerelle intermédiaire pour les travaux de courte durée. Il sauvegarde toutes les échantillons récupérées localement, et exécute des règles sur ces données afin d'agréger et d'enregistrer de nouvelles séries chronologiques à partir de données existantes ou de générer des alertes.

Ensuite, GRAFANA ou d'autres interfaces graphiques sont utilisées pour visualiser les données collectées.

PROMETHEUS enregistre toutes les séries chronologiques purement numériques. Il convient à la fois, à la surveillance centrée sur la machine, et à la surveillance d'architectures très dynamiques orientées services. Dans un monde de micro services, sa capacité à prendre en charge la collecte de données multidimensionnelles, est un atout particulier.

PROMETHEUS est conçu pour la fiabilité, pour être le système auquel on s'adresse pendant une panne pour nous permettre de diagnostiquer rapidement les problèmes. Chaque serveur PROMETHEUS est autonome et ne dépend pas du stockage réseau ou des autres services distants. On peut s'y fier lorsque d'autres éléments de notre infrastructure sont en panne, aussi, on n'a pas besoin de configurer une infrastructure étendue pour l'utiliser.

4- Fonctionnement de PROMETHEUS

PROMETEHUS est un concept de récupération et stockage de données (Pull/store), c'est à dire cueillir les métriques dans des cibles prédéfinies à travers un port spécifique.

5- Exportateurs (EXPORTER)

Les exportateurs sont des serveurs de métriques compatibles (Clé/valeur) par PROMETHEUS, c'est-à-dire, de type : Gauge et compteur.

Il existe plusieurs panoplies d'exportateurs, dont le plus utilisé est le Node-Exporter qui est un exportateur de métriques, provenant des composants physiques d'un serveur ou dans un nœud d'un réseau (CPU, Mémoire, Transmission, Réception…). En gros, PROMETHEUS cible ces exportateurs pour collecter les métriques qui y sont exposés. La figure suivante illustre le flux de récuperations de metriques sur Node-Exporter.

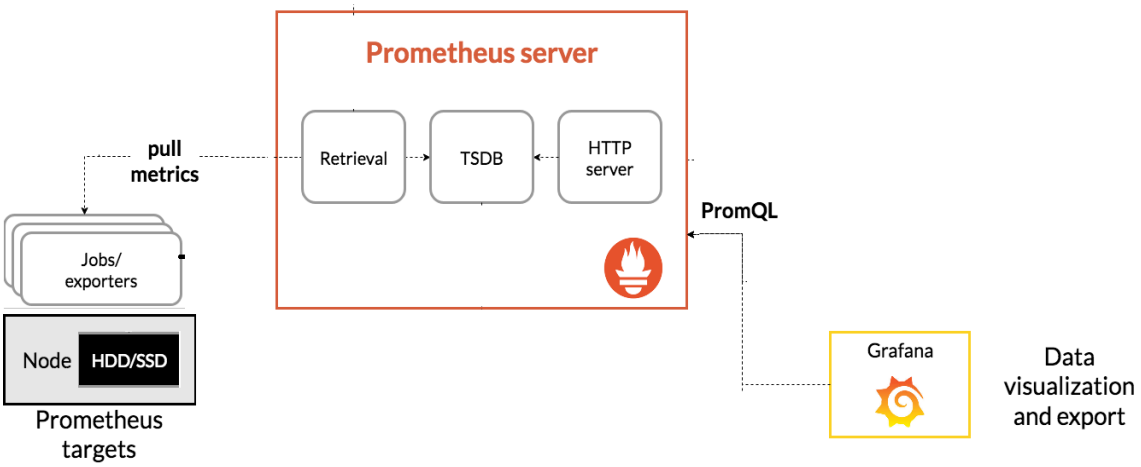

Figure 37: Node-Exporter et récupération des métriques

6- PUSHGATEWAY

Comme PROMETHEUS ne collecte que les métriques exposés à un instant précis (t), il se peut que des métriques correspondantes à un temps **t+1** ne soient plus exposées notamment pour les tâches de très courtes durées. Il faut noter qu'avec PROMETHEUS, les collectes se font après chaque période définie et généralement toutes les 5 ou 15 secondes, selon les besoins.

Pour pallier à ces pertes de métriques, alors utiles, notamment pour les tâches à très courtes durées, PUSHGATEWAY a été mise à disposition afin d'y envoyer ces données.

PUSHGATEWAY est donc disponible, justement, pour les tâches à très courtes durées. Un tel processus de renvoi est illustré par la figure suivante.

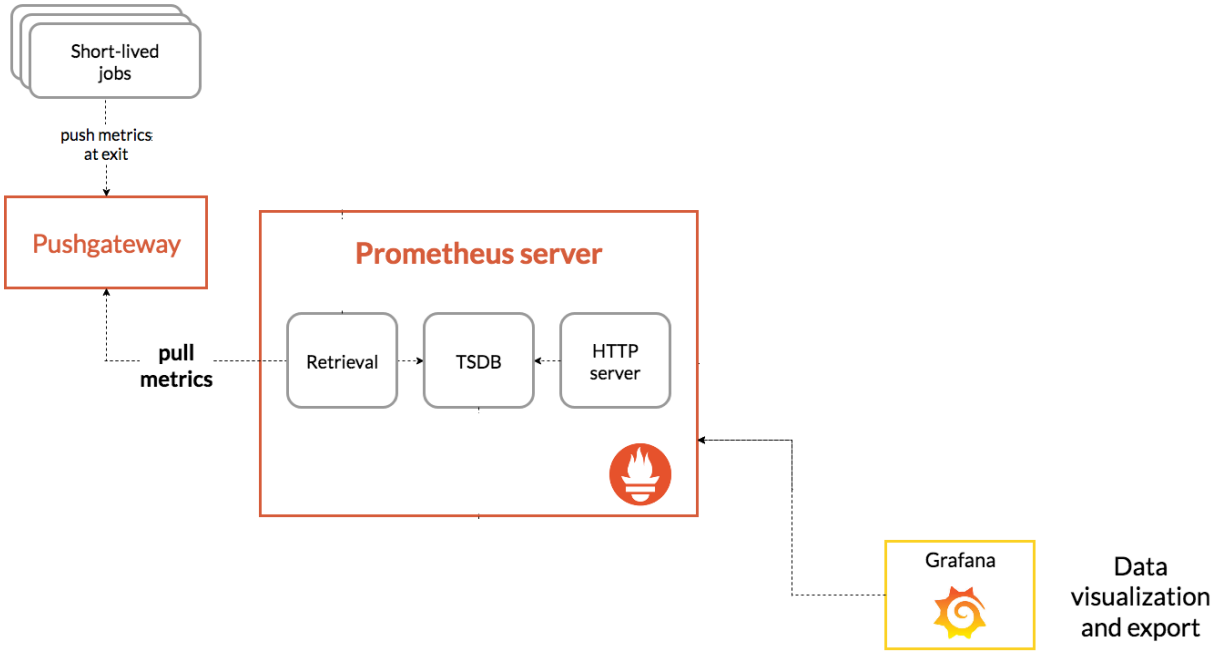

Figure 38: Tâches à durée courtes et poussage des métriques vers PUSHGATEWAY

7- PROMETHEUS en tant que serveur

PROMETHEUS peut jouer le rôle d'un serveur, où les utilisateurs peuvent envoyer des requêtes (PromQl) pour accéder aux informations dans la base données de type temps/séries.

8- PromQL

C'est un langage de requête spécifique à PROMETHEUS se nommant PromQL (PROMETHEUS Query Language). On l'utilise pour formuler des requetés à la base de données de PROMETHEUS.

9- Gestion d'alertes :

PROMETHEUS a des fonctionnalités qui peuvent gérer les alertes pour des situations prédéfinies, et dont les valeurs collectées diffèrent des références. Ces mécanismes sont, généralement, utiles dans des pics de transmission, de réception, ou dans des pannes.

10- Lancement et configuration de PROMETHEUS

- Commande :
  cd ~/prometheus-2.7.1.linux-amd64/ ; ./prometheus --config.file=promnode.yml
  Serveur disponible sur localhost : 9090
- Configuration de la cible dans le fichier promnode.yml
- Lancement et ciblage de Node-Exporter
  Commande : cd ~/node_exporter-0.17.0.linux-amd64/ ; ./node_exporter

Annexe C : GRAFANA

GRAFANA est un concept ouvert qui permet d'analyser graphiquement des métriques stockées dans un serveur, ou d'une base de données. L'analyse consiste à interroger, à visualiser, à comprendre, et d'alerter.

Grâce à ces fonctionnalités, on peut avoir un tableau de bord intelligent et des systèmes d'alertes optimaux.

Dans notre projet, GRAFANA est l'outil dont nous avons choisi pour analyser graphiquement les données stockées dans PROMETHEUS

1- Visualisation

Des cartes et des histogrammes sont disponibles aux utilisateurs pour permettre de visualiser et de comprendre les données recensées.

Avec GRAFANA, il existe plusieurs types de graphes disponibles :

Gauge, tableau statistique, histogramme, compteur, etc…

2- Unification

GRAFANA supporte différents types de données, notamment le type de données de PROMETHEUS de type temps/série. Les utilisateurs ont la possibilité d'unifier leurs consoles d'analyse dans une même plateforme de visualisation.

3- Alertes

Personnaliser des alertes graphiques est l'une de fonctionnalités de GRAFANA, ce qui facilite les suivis des anomalies, des surcharges de trafic ou autres.

4- Lancement et configuration GRAFANA

- Commande : cd ~/grafana-6.0.0.linux-amd64/grafana-6.0.0/bin/ ; ./grafana-server
- Serveur disponible sur localhost:3000.

La figure suivante nous montre un tableau de bord sur GRAFANA

Figure 39: Exemple de tableau de bord sur GRAFANA