PROJET-SDN

Par **LEBEL Alexis** et **LEFEVRE Elwan**

A destination de **HUSSENET Laurent**

DUT RT Chalons en Champagne

Année 2021-2022

RÉSUMÉ

Résumé

Dans ce rapport, nous allons explorer la création d'une interface de communication pour les actifs réseau des salles de TP de l'IUT. Cette interface sera un moyen de créer ou de modifier une infrastructure réseau de façon dynamique, en fonction des besoins du professeur. Nous parlerons du contexte derrière le projet, c'est à dire de l'infrastructure déjà en place, des besoins qui nous ont mené à mener ce projet et des solutions possibles explorées. Nous traiterons ensuite les différentes étapes de la création de l'interface, en commençant par la connexion entre l'api et le switch, puis de la conception de l'api en elle même, de la conception de l'interface web et de l'aspect de travail collaboratif. Enfin, nous examinerons les améliorations envisageables, en commençant par la mise à jour de l'interface web en temps réel, puis la découverte automatique de l'infrastructure et l'instanciation auto d'apis, puis la prise en charge de switch de nouvelle génération, et enfin la création de graphique et d'alerte pour de la supervision.

Mots clés

- **SDN - Software Defined Network** : Concept de réseau défini par le logiciel. Cela permet une grande facilité de déploiement et de développement de l'infrastructure réseau.
- **Cisco** : Fabricant d'actifs réseau, leader du marché.
- **API - Application Programming Interface** : Interface de programmation d'application. C'est une interface de communication entre l'application et l'actif réseau.
- **SNMP - Simple Network Management Protocol** : Protocole de supervision basée sur un système d'OIDs (Object Identifier).
- **Flask - Microframework Python** : Framework de développement web basé sur Python.

- **FastAPI** : Framework de développement web basé sur Python. Permet de développer des APIs facilement et avec une documentation générée automatiquement.
- **Docker** : Logiciel de déploiement de conteneurs. Permet de déployer des conteneurs sur des serveurs. Les conteneurs sont des applications qui fonctionnent dans un environnement complètemment isolé du reste du système. Cela permet une gestion sous forme d'unités indépendantes, de façon à pouvoir les gérer indépendamment.
- **GCP - Google Cloud Platform** : Plateforme Cloud de Google. Ici, nous nous en servons comme plateforme d'hébergement de conteneurs.
- **Github** : Plateforme de développement collaboratif. Permet de partager des codes, de créer des dépôts et de gérer des dépôts.
- **Hugo** : Site de publication de contenus. Permet de générer des sites web statiques à partir de contenus écrits en Markdown.

REMERCIEMENTS

Pour commencer ce rapport de projet, nous tenions à remercier notre tuteur technique **M. HUSSENET Laurent** pour son soutien et son aide tout au long du projet.
Un grand merci également à notre tutrice **Mme. GOBILLARD Delphine** qui nous a aidé à nous organiser et préparer les differentes épreuves.
Nous remercions de manière plus générale toutes l'équipe pédagogique qui nous a soutenu tout au long de ce projet.
Enfin, nous remercions **Alexandre GOSSARD** et **Corentin FREIRE**, avec qui nous nous sommes entraidés dans la réalisation de nos projets respectifs. C'est grâce à eux que nous avons pu développer une compétence essentielle dans la réalisation de nos projets futurs : l'entraide au delà des frontières de l'équipe.

TABLE DES MATIÈRES

Introduction

1. Idée Générale
    1. Contexte du projet
    2. Attentes du projet
2. Découpage fonctionnel de l'application
    1. Qu'est-ce qu'une topologie
    2. Le portail web
    3. Gestionnaire de topologies
    4. Base de données de topologie
3. Implémentation du projet
    1. Portail web
    2. Interface de configuration logicielle
    3. Base de données
    4. Gestionnaire de toplogies
4. Mise en oeuvre du projet
    1. Ossature du logiciel
    2. Outils de collaboration
    3. Évaluation

Conclusion

INTRODUCTION

Le **SDN** ou **Software Defined Network** est un concept dont le but est de créer ou modifier une infrastructure réseau de façon logicielle. Ce concept est un des sujets de projet qui nous ont été proposés par l'IUT de Châlons-en-Champagne, dans le cadre du projet de deuxième année de DUT Réseaux et Télécommunications. Nous avons choisi ce sujet car il est très formateur et ouvre des perspectives d'acquisition de compétences dans un spectre très large, très utiles pour le développement d'un projet professionnel. Ce projet nous a aussi permis de développer des compétences en programmation, en supervision, en gestion de projet et réseaux. Nous espérons ainsi pouvoir utiliser ces compétences dans notre future carrière.

Sur les actifs de nouvelle génération, le concept de SDN est integré dans les systèmes d'exploitation des actifs réseau, sous forme d'une **interface de communication**, reposant sur des technologies de communication standards (HTTP). Cette interface appellée **API** pour Application programming Interface, permet la configuration des actifs en envoyant des requêtes HTTP à l'API, c'est à dire, des requêtes similaires à celles utilisées par un navigateur web. Les actifs d'ancienne génération en production dans les salles de Travaux Pratiques, à l'IUT, sont dépourvus de cette interface. Il est donc nécessaire de **créer une interface de communication** pour ces actifs réseaux.

Ce constat nous ammène à la problématique suivante : **Comment adapter facilement la configuration des actifs réseau des salles de TP ?**

Nous commencerons tout d'abord par exposer certaines **généralités** concernant le projet, puis nous parlerons plus généralement du **découpage fonctionnel** de notre application. Nous enchainerons ensuite sur deux parties plus techniques concernant la **réalisation** du projet et sa mise en œuvre en **production**.

GÉNÉRALITÉS

CONTEXTE DU PROJET

L'IUT de Chalons-en-Champagne propose un enseignement axé sur la **pratique**. Dans ce but, des salles de TP sont mises à la disposition des étudiants, celles-ci équipées de matériel réseaux.

Des services à déployer

Les différents TP nécessitent de déployer différents **services** adaptés aux différents besoins du TP. Ces différents services, sont des extensions de ceux présents dans le Data-Center. Concrètement, ces services sont projetés sur les prises réseaux à l'avant du switch dans chaque salle de TP de façon fixe et manuelle.
Ils **dépendent de la promotion et du groupe** utilisant la salle, ainsi que de la **nature** du TP et peuvent être par exemple, l'accès à un réseau privé de binôme.

La projection de ces services à ce jour

A ce jour, la projection de ces services est dite **statique** ce qui signifie que tous les cas sont manuellement projetés dans chaque salles. Cette façon de faire implique un manque de prise réseaux pour la projection de services supplémentaires dans des cas spécifiques. Dans le cas de TPs nécessitant des services non-disponibles dans la salle de TP, il faut redéployer **manuellement** des services, et les personnes abilitées à le faire sont peu nombreuses. Dans le cas d'un TP mené par un intervenant extérieur avec des besoins spécifiques, cela nécessite l'action d'une personne abilitée, pas forcément disponible.

ATTENTES DU PROJET

Le projet consiste à développer un outil permettant de gérer la **projection** des services dans les salles de TP. Cet outil, de préférence sur une plateforme **mobile**, sera accessible par toutes les personnes disposant de l'accès au réseau de l'IUT via le service de tunnel **VPN** déjà en place.

Le placement dans les salles de TP d'un accès via **QR code** sera également possible, afin de faciliter l'accès au service. Le changement de **topologie** devra être simple et rapide.

Ce que l'on appelle une topologie est un **ensemble de projections de services** sur des prises réseaux. Cela se manifeste par la modification de la configuration des prises réseaux dans les salles de TP. Les **paramètres** qui pourront varier sont multiples : mode de projection (accès à un seul service, envoi de plusieurs services (trunk)) et ports en fonctionnement.

Ces paramètres, afin de former une topologie, seront définis dans un texte de configuration, au format YAML, c'est à dire un fichier dont le contenu consiste en un assemblage de **couples clé/valeur**, facilement lisible et modifiable. Exemple : `clé: valeur`

Cet outil devra proposer une méthode simple et graphique permettant **l'ajout, la modification et la suppression** des topologies, tout en permettant de **visualiser** les différentes projections.

Situation initiale et prérequis :

- M. BATISTE, professeur de l'IUT de Chalons-en-Champagne, enseigne un module d'infratructure wifi.
- La configuration des prises réseaux est conçue pour un usage général pour la plupart des TP.
- Ce TP requiert une configuration spécifique, pour avoir accès aux services du data-center.

- Le système est déployé sur le data-center, et un QR code est affiché à l'entrée des salles de TP.

Que doit-on faire pour déployer une topologie adaptée aux besoins du TP ?

- M. BATISTE, en arrivant dans la salle de TP, se connecte au VPN et scanne le QR code.
- Il arrive sur l'interface web liée à l'infrastructure de la salle.
- Il se connecte, et arrive sur une page récapitulative.
- De là, il peut choisir la topologie à utiliser, puis l'appliquer depuis l'interface WEB.
- Dans l'éventualité où la topologie choisie n'est pas disponible, il peut configurer manuellement les prises réseaux.
- Il peut ensuite sauvegarder la topologie ainsi créée.

DÉCOUPAGE FONCTIONNEL DE L'APPLICATION

QU'EST-CE QUE UNE TOPOLOGIE ?

L'intégralité de ce projet se base sur la capacité à déployer facilement des topologies variées sur les actifs réseaux. Nous allons donc voir ce qu'est une topologie et comment les définir.

Une topologie est donc un fichier de configuration que nous pouvons déployer sur les switchs. Ce fichier contiendra la configuration des différentes interfaces sous un format défini au préalable (JSON, YAML ...).

La configuration en elle même, spécifie les configurations d'interfaces (**trunk**, **access** ou encore désactivée), on y trouve également les informations relatives aux différents **VLANs** (noms et numéros). La topologie est donc un empilement de l'ensemble de ces informations.

La sélection de la topologie se fera par la suite au travers de l'interface web dans laquelle nous devrons définir les critères de choix de la topologie.

Pour cela on se base sur quatre critères principaux :

- La salle de TP
- La classe (RT1, RT2 ...)
- Le groupe de TD/TP (TP1A, TP1B ...)
- Le TP (TP wifi, TP téléphonie ...) Basé sur ces quatre critères on pourra définir la topoplogie souhaité.

Une fois la topologie choisie elle sera traduite dans un format adapté pour les actifs réseaux par le gestionnaire de topologie.

LE PORTAIL WEB

Le **portail web** est la partie visible du projet, c'est notre interface utilisateur. Étant en contact avec l'utilisateur, notre interface doit donc être **facile d'accès** et facile d'utilisation.

L'accès à l'interface se fera de plusieurs façons pour la rendre la plus accesible possible. La première façon est via son URL (www.sdn.chalons.univ-reims.fr). La seconde est l'utilisation d'un **QR Code** placé dans les différentes salles de TP ainsi l'utilisateur aura juste à scanner le code pour arriver sur l'interface web où l'utilisateur devra s'identifier avant de choisir la topologie qu'il souhaite déployer.

La gestion des droits et le système d'authentification est défini par un couple identifiant/mots de passe pour les étudiants et les professeurs. Ceux-ci ne sont pas individuels et sont sauvegardés dans un **fichier de configuration**.

L'interface en elle même, afin de rendre l'expérience utilisateur la plus simple et agréable possible, a été principalement développée pour fonctionner sur **mobile**. Cette interface se compose alors de quatres onglets.

Le premier onglet contiendra une vue "dashboard", qui est un **résumé** de la salle actuellement séléctionnée, on y trouvera par exemple le statut des actifs pour connaître leur état de fonctionnement.

Le second, "application de la configuration" est dédié à l'application de topologie. C'est sur cette page que l'on pourra choisir la topologie que l'on souhaite déployer ainsi que visualiser quelle topologie est actuellement en production. On pourra, en sélectionnant la classe, le groupe et le TP, choisir la configuration adaptée et la mettre en production.

Le troisième onglet, "création de configuration" a lui pour but de créer les différentes configurations. Pour cela, on est ammené à sélectionner les différents **vlans**, **trunk** et **interfaces** que l'on peut alors configurer manuellement avant de sauvegarder cette configuration pour la déployer ultérieurement.

Le dernier onglet "configuration On The Go" a pour fonction d'activer et désactiver les différentes interfaces avec un clic. Cela permettra donc de facilement gérer le statut des interfaces.

GESTIONNAIRE DE TOPOLOGIE

Le gestionnaire de topologise est la pièce maitresse de notre projet, c'est la partie logicielle qui fait le **lien entre le portail web, les actifs ainsi que la base de données**. Nous allons voir concrètement comment il fonctionne et pourquoi nous en avons besoin.

Aujourd'hui change une configuration s'effectue de manière manuelle en utilisant des protocoles qui nécéssitent une intervention humaine (**port console**, **ssh**, **telnet**), cependant cela génère une dépendance humaine. Car même si ces méthodes permettent de modifier les configurations, cela requiert une intervention d'une **personne abilitée**.

Pour palier ce problème, les switchs les plus récents disposent d'interfaces de programmation logicielles, aussi appelé **APIs**, mais cette technologie n'est disponible que sur les modèle les plus récents et donc n'est pas accesible sur tous le **parc d'actifs** de l'IUT.

Sur des modèles plus anciens, il est néanmoins possible de lire et de modifier des configurations en utilisant des **outils de supervision**.

Notre gestionnaire de topologies aura donc pour but de rendre **compatible** ces deux générations de matériel.

Pour cela nous avons donc envisagé de construire cette interface logiciel afin de permettre aux **anciens modèles** d'équipement tout comme aux plus récents d'être configurés depuis une seule et même interface.

Ce gestionnaire de topologies est également le pont entre notre portail web, l'API que nous développons et notre base de donnée car en effet c'est au travers de celui-ci que nous allons être capable de projeter et récupérer les topologies sur les actifs, grâce à une interface **Web**.

BASE DE DONNÉES DES TOPOLOGIES

La base de données est la mémoire de notre projet. C'est l'outil qui va nous permettre de stocker l'intégralité des différentes topologies, mais, également de les retrouver quand c'est nécessaire. Les fichiers de topologies peuvent ressembler à ceci :


```
{
  "name": "Vlan1",
  "description": "Vlan1",
  "port_id": 1,
  "status": "up",
  "operstatus": "up",
  "vlan": null,
  "speed": 1000000000
},
{
  "name": "Vlan10",
  "description": "Vlan10",
  "port_id": 10,
  "status": "up",
  "operstatus": "up",
  "vlan": null,
  "speed": 1000000000
},
{
  "name": "StackPort1",
  "description": "StackPort1",
  "port_id": 5179,
  "status": "up",
  "operstatus": "up",
  "vlan": null,
  "speed": 0
}
```

(cf Annexe 5)

C'est donc ainsi que les informations nécessaires à la configuration seront stockées dans notre base de données.

Mais ces informations peuvent venir sous trois formes :

La première est dans le cas de la configuration d'un vlan. Dans ce cas, chez cisco il est nécessaire de faire :

```
SWITCH(config)#vlan {numero}
SWITCH(config-vlan)#name {name}
```

Ce qui dans notre base de données ce résumera par :

```
{
  "description": "{name}",
  "dot1q_id": {numero}
}
```

Le second cas est celui d'une interface en mode **trunk**, ce qui nécessite de faire :

```
SWITCH(config)#interface fastethernet 0/1
SWITCH(config-if)#switchport mode trunk
```

Dans notre cas, on obtiendra :

```
{
  "interface": {
    "name": "FastEthernet2/0/47",
    "description": "FastEthernet2/0/47",
    "port_id": 10547,
    "status": "up",
    "operstatus": "down",
    "vlan": null,
    "speed": 100000000
  },
  "native_vlan": {
    "description": "RT2G1B6-0",
    "dot1q_id": 415
```

```
    },
    "tagged_vlans": [
      0,
      4,
      8,
      16,
      26,
      351,
      352,
      353
    ],
```

Le dernier cas que nous pouvons rencontrer est le cas d'une interface en mode **access** qui se traduit en dialecte Cisco par :

```
SWITCH(config)#interface ethernet0/1
SWITCH(config-if)#switchport mode access
```

Ce qui donnera dans la base de données :

```
  {
    "name": "FastEthernet1/0/38",
    "description": "FastEthernet1/0/38",
    "port_id": 10038,
    "status": "up",
    "operstatus": "down",
    "vlan": {
      "description": "rt-reseau-1",
      "dot1q_id": 64
    },
```

Toutes les configurations que nous enregistrons dans la base de données, ne sont pas à faire manuellement, il est possible de récupérer une configuration qui est déjà en production ou encore dans créer une nouvelle au travers de l'interface web, ce qui rend la création bien plus simple.

RÉALISATION DU PROJET

PORTAIL WEB

Le portail web est la partie en contact avec l'utilisateur. Elle doit permettre à l'utilisateur d'interagir avec le système. Ce portail web est composé de plusieurs parties :

Partie serveur : Flask

Cette partie est celle qui tourne sur le serveur. Elle gère les requêtes HTTP générées par l'utilisateur. Cela veut dire que l'utilisateur peut accéder à des pages, qui sont générées par le serveur. Le serveur qui met à disposition le portail web est appellé **Werkzeug**. Il s'agit du serveur integré au paquet **Flask**, de Python. Ainsi, pour lancer le serveur, il suffit de lancer le script Python qui contient le point d'entrée du serveur.

La partie serveur repose aussi sur **Jinja2**, qui est un moteur de template. Il permet de générer des pages HTML à partir de templates. Les templates sont des fichiers HTML qui contiennent des instructions pour y intégrer des données de façon dynamique. (cf. Annexe 1)

La partie authentification est gérée par le module **flask-login**. Il permet de gérer les connexions de l'utilisateur. Les informations de connexion sont stockées dans un fichier YAML qui est lu par le module **flask-login**.

La finalité de la partie serveur est de fournir les pages WEB à l'utilisateur, et de génerer les requêtes à l'interface logicielle (API) de l'actif à gérer en fonction des demandes de l'utilisateur. Concrètement, lorsque l'utilisateur veut changer un paramètre, il clique sur un bouton, et le serveur génère une requête HTTP qui est envoyée au module logiciel. Le module logiciel gère la modification du paramètre, et renvoie une réponse HTTP qui est envoyée au serveur. Le serveur renvoie la réponse à l'utilisateur. Ainsi, le serveur agit comme un intermédiaire entre l'utilisateur et le module logiciel, ce qui rend transparent l'utilisation du module logiciel.

Partie client : Javascript

Cette partie est celle qui est gérée par le navigateur. Les **scripts** Javascript sont plutôt simples, et sont principalement dédiés à des fonctions de **transformation** de la page. Concrètement, il s'agit de la barre de navigation qui disparait et réapparait lorsque l'utilisateur clique sur un bouton, sur mobile, ou encore la configuration qui apparait dans le champ de texte lorque l'utilisateur clique sur l'élément de configuration dans la partie "Configuration". Cette partie est gardée volontairement simple, puisque le gros du travail est géré par le serveur Flask.

État d'avancement

A ce jour, le portail web est en cours de développement. Voici les fonctions qui sont actuellement implémentées :

- Connexion
- Déconnexion
- Résumé
- Création de configuration
- Modification de configuration
- Application de configuration
- Configuration "A la volée"

Il est possible d'ajouter la mise à jour automatique de la page Résumé lorsqu'un port change d'état (Branché, débranché). Cela permet de voir rapidement les changements de port. Cependant, par contrainte de temps, nous n'avons pas pu implémenter cette fonctionnalité. Nous y avons quand même pensé, et cela nous a permis de mettre au point le fonctionnement général et un exemple de code (cf Annexe 2). Ci-dessous, un schéma du fonctionnement de cette fonctionnalité.

INTERFACE DE CONFIGURATION LOGICIELLE.

L'interface de configuration logicielle est un service permettant la configuration des actifs réseaux.

Pourquoi cette interface ?

Tout d'abord, il est important de savoir comment les actifs réseaux sont configurés. Nous avons vu dans la section précédente qu'il était admis que nous n'utiliserions pas de protocoles interactifs comme le port Console ou le SSH (Console via réseau). Il a donc fallu trouver d'autres moyens. Nous nous sommes arrêtés sur l'utilisation d'un protocole bien connu dans le domaine de la supervision : le protocole **SNMP**. Il s'agit d'un protocole qui permet de communiquer avec tout type d'appareils (ordinateur, switch, routeur, etc.) afin de récupérer des informations, ou de pousser de la configuration. Cette interface a donc pour but de traduire les actions SNMP en URL paramétrables. Cela permet une utilisation du protocole assez transparente et facile.

Le protocole SNMP

Le protocole SNMP, basé sur **UDP**, est un protocole qui permet de communiquer avec des appareils pour récupérer des informations ou pousser de la configuration. Il utilise un système d'**OID** (Object Identifier) pour définir les informations que l'on souhaite récupérer. Un OID est un identifiant unique qui permet de définir un objet. Il est défini sous la forme d'une suite de chiffres, séparés par des points. Exemple d'OID : `1.3.6.1.2.1.1.5`

(Permet de récupérer le nom d'hôte). Les OIDs nécessaires pour notre interface sont définis dans le fichier de configuration de ce service (cf Annexe 3).

Configuration obligatoire de l'actif réseau

Il est important de configurer l'actif réseau avant de pouvoir utiliser l'interface de configuration sur ce dernier. Voici les paramètres à configurer :

- **Interface SVI** : l'interface SVI de l'actif réseau. C'est une prise réseau virtuelle qui est utilisée pour configurer l'actif réseau.
- **Adresse IP** : l'adresse IP de l'actif réseau.
- **Communauté SNMP** : la communauté SNMP de l'actif réseau. C'est une valeur textuelle qui définit la clé de sécurité qui permet la configuration de l'actif réseau.

Une API :

Puisque ce service traduit les actions SNMP en URL paramétrables, il s'agit d'une **API**. Une API est un serveur qui permet, via des URL, de récupérer des informations ou d'exécuter des actions. Dorénavant, nous ne parlerons plus d'"interface de configuration logicielle" mais d'**API**. Une liste des toutes les URLs de l'API est disponible en Annexe 4.

Construction de l'API

Open API

Afin de nous rapprocher au maximum du côté "Terrain", nous avons décidé de construire notre API suivant le **standard Open API**. C'est un standard qui permet de définir des structures de données, et de ressources. Il définit des méthodes HTTP telles que GET, POST, PUT, DELETE. Ces méthodes ont toutes leurs fonctions. Par exemple, une requête GET permettra de récupérer des informations, et une requête POST permettra de pousser des informations.

Le paquet Python que nous utilisons pour construire notre API est **FastAPI**. C'est un framework (Littéralement "Cadre de travail" -> Un ensemble d'outils) qui permet de

construire rapidement une API. L'avantage de ce paquet, est qu'il génère automatiquement la documentation de l'API, en respectant le standard Open API.

Support du SNMP en Python

L'utilisation du protocole SNMP en Python est prise en charge par deux paquets différents :

- **snmp-cmds** : Ce paquet est responsable de récupération de tables complètes (Table des interfaces, etc.) via le protocole SNMP. Il gère aussi toute la partie de poussée de configuration.
- **pysnmp** : Ce paquet prend en charge ce que le premier paquet ne peut pas. Il permet de récupérer des informations qui ne sont pas dans des tables.

Le cache

La récupération initiale de toutes les données est très longue (environ 7 secondes), et cela est lié au fonctionnement même du protocole SNMP. Pour éviter de devoir récupérer les données à chaque fois, nous avons décidé de créer un **cache**. Ce cache est géré par le paquet Python **Beaker**. Il s'agit d'un cache stocké en RAM, et qui permet de stocker des données pendant une durée définie. Ce paquet est tout indiqué par rapport au paquet **Redis**, car les informations stockées ont une taille définie qui ne bouge pas ou peu, on ne risque donc pas de dépassement de RAM. Redis, lui, est un cache beaucoup plus puissant, mais aussi beaucoup plus compliqué. Sa puissance aurait été énorme par rapport au travail qui lui aurait été demandé, donc il n'était pas rentable de l'utiliser.

Authentification

Pendant le développement de notre API, nous somme partis sur une authentification au niveau de celle-ci. Cette authentification était basée sur une collection d'utilisateurs dans la base de données, avec un système de token **JWT**. Concrètement, l'utilisateur fournit un nom d'utilisateur et un mot de passe, et nous vérifions que ces informations sont correctes. Si elles sont correctes, nous générons un jeton, un code, qui doit être inclus dans les requêtes HTTP, pour celles qui sont protégées.

Cependant, avec le recul, nous nous sommes rendu compte que nous avions été chercher trop compliqué, puisque dans le cahier des charges, il était stipulé que l'API n'allait pas être **exposée**. Ainsi, l'authentification à ce niveau n'était pas nécessaire. Nous avons donc décidé de ne pas utiliser d'authentification, et de laisser l'ensemble de l'API **accessible**, sachant que seul le service de portail web aura accès à elle. L'authentification sera donc

effectuée par le service de portail web, à partir d’infomations contenues dans le fichier de configuration de ce service.

État d’avancement

A ce jour, on considère l’API comme terminée. Lors du développement de nouvelles fonctionnalités de l’interface web, il sera peut être nécessaire de la compléter, mais pour l’instant, l’API réalise les fonctions nécéssaires pour le bon fonctionnement du projet.

BASE DE DONNÉES

Structure des topologies

Les topologies sont stockées sous forme de chaînes de caractères au format YAML. La structure de cette chaine est simple, elle est composée de 4 parties :

- **name** : le nom de la topologie -> utilisé pour identifier la topologie à la place de son ID, beaucoup trop lourd pour l'utilisateur
- **vlans** : la liste des VLANs utiles à l'application de la topologie, à créer si nécessaire
- **interfaces** : la liste des configurations des interfaces de la topologie
- **trunks** : la liste des configurations des trunks de la topologie

Un exemple de topologie est proposé en annexe 5

Comment sauvegarder une topologie

Tout d'abord, il a fallu faire un choix au niveau de la technique de stockage de ces données. Nous avions plusieurs solutions :

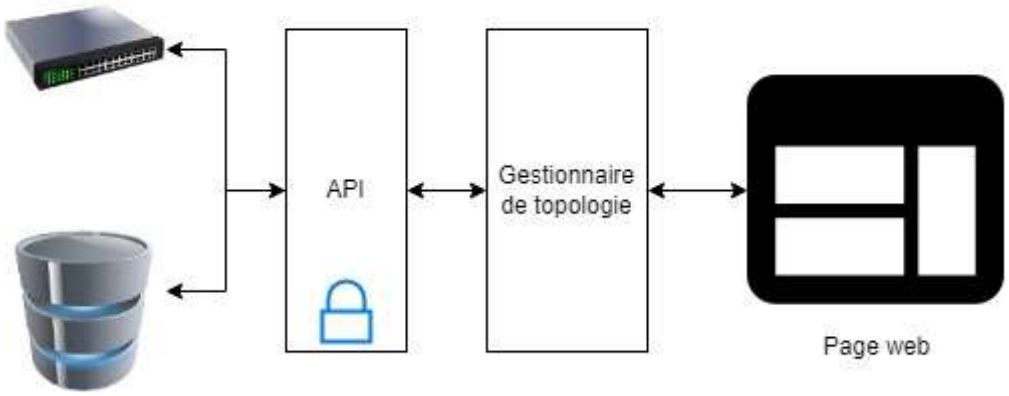
- Les stocker sous forme de fichiers **YAML**, dans un dossier spécifique sur le serveur
- Les stocker dans une **base de données**

Nous avons choisi l'option de la base de données, mais là encore, beaucoup de choix se sont offerts à nous. Lors de notre cursus à l'IUT, nous avons étudié le système de base de données **PostGreSQL**, qui est un système relationnel, c'est à dire que les données sont stockées dans des tables, qui sont liées entre elles. Cela impose un format de données assez **statique**, sans compter la conception de la structure de la base. Nous sommes donc partis sur une base capable de stocker des données au format clé/valeur (JSON),

MongoDB. MongoDB nous permet de stocker nos topologies quasiment tel quel, sans devoir les formater, pour les faire rentrer dans des tables. L'avantage de ce système, c'est qu'**à un identifiant, est lié une topologie**, ce qui est bien plus simple pour la gestion de la base, mais aussi les lectures et écritures dans celle-ci.

Interactions avec le gestionnaire de topologie

Lorsque nous avons commencé le projet, nous sommes partis sur une API, gérant l'ensemble des opérations liées à la base de données, à l'actif, et à l'authentification. Le système que nous avons développé n'implique donc pas une communication directe de la base de données avec le gestionnaire de topologie, mais uniquement avec l'API, qui **interface** la base et la partie interface utilisateur.



La logique derrière ce choix, était de se rapprocher le plus possible de l'architecture des logiciels en production aujourd'hui.

Néanmoins, avec le recul et grâce aux discussions que nous avons eu avec notre tuteur de projet, nous nous sommes rendu compte que nous avons ajouté de la **complexité** au projet, et qu'il aurait été plus simple de faire dialoguer le gestionnaire de topologie directement avec la base de données. Ce choix concorde malgré tout, avec celui de déléguer la responsabilité de l'application des topologies à l'API. C'est pour cela que nous n'avons pas fait marche arrière et que nous sommes restés sur cette architecture.

Stockage des utilisateurs

A l’origine, les utilisateurs aussi, étaient stockés sur la base de données. Elle était consultée par l’API, qui gérait l’authentification. Cette architecture était assez **complexe**, et nous avons donc décidé de ne pas la conserver, puisque nous nous sommes encore une fois rendu compte que nous avions **complexifié** le projet. Nous avons donc décidé de stocker les utilisateurs dans un fichier YAML, et de déléguer la connexion des utilisateurs au gestionnaire de topologies. Cela implique aussi de rendre l’API ouverte, sans que l’authentification soit nécessaire, mais cela n’est pas handicapant, puisqu’à terme, celle-ci ne sera pas exposée.

GESTIONNAIRE DE TOPOLOGIES

Le gestionnaire de topologies est l'élément central de notre projet. C'est lui qui fait le lien entre l'**utilisateur** via l'interface web, et les **actif réseaux** via l'**API**. C'est l'équivalent du contrôleur, dans les modèles de programmation MVC (Model-View-Controller).

Interactions

Avant toute chose, l'utilisateur doit se **connecter** à l'interface web. Il se rend donc sur la page d'accueil du portail web. Il y trouve un bouton qui permet de se connecter. Une fois ses identifiants rentrés, l'appui sur le bouton de connexion envoie une requête au gestionnaire qui lui autorise ou non l'accès à l'interface de configuration.

Via l'interface web, l'utilisateur peut choisir la **topologie** qu'il souhaite déployer sur les actifs réseaux. Il s'agit d'une des tâches de ce gestionnaire. Concrètement, le gestionnaire vient demander à l'API les topologies disponibles, et les affiche dans une liste, dans l'onglet "**Configuration**". L'utilisateur peut alors choisir la topologie qu'il souhaite déployer sur les actifs réseaux. Une fois validée, le gestionnaire va demander à l'API de déployer la topologie choisie de façon **autonome**.

L'utilisateur peut aussi choisir de configurer l'actif **manuellement**. Il s'agit d'une autre tâche de ce gestionnaire. concrètement, il vient demander à l'API de l'actif, l'état de tous les ports, et les affiche dans une **liste en forme de switch**. L'utilisateur peut alors choisir les ports qu'il souhaite configurer, et changer les valeurs des paramètres. Une fois validée, le gestionnaire va demander à l'API de configurer l'actif, selon les consignes de l'utilisateur, de façon **unitaire**.

Une fonction, que nous n'avons pas encore développée, permettra de mettre à jour de façon **dynamique** l'affichage de l'interface lorsque un appareil est branché ou débranché sur un des ports du switch.

Authentification

Comme dit précédemment, nous avions à la base, développé un système d'authentification au niveau de l'API, avec les couples identifiant/mot de passe stockés dans la base de données. Cependant, nous avons décidé de ne pas utiliser cette méthode, car elle était beaucoup trop compliquée par rapport au cahier des charges que nous avions. Nous avons donc décidé de créer un système d'authentification au niveau du gestionnaire, avec les identifiants/mot de passe stockés dans un fichier de configuration. Cela nous donne donc deux comptes :

- Un compte **administrateur**, qui a les droits d'administration du gestionnaire : pour les enseignants.
- Un compte **utilisateur**, qui a les droits de lecture seule : pour les étudiants.

Différences par rapport à ce qui a été pensé à la base

Au départ, il était prévu que le gestionnaire de topologie soit responsable de créer et appliquer les configurations sur les actifs, de façon unitaire. Concrètement, à partir des configurations, il devait demander à l'API d'effectuer les changements un par un sur les actifs. Cependant, nous avons décidé de déléguer cette tâche à l'API. En effet, les **latences** inhérentes au protocole de supervision SNMP sont trop importantes pour un traitement unitaire, depuis un système externe à l'API. C'est donc elle qui gère l'application des configurations sur les actifs, en renvoyant les directives via **SNMP** lorsque celles-ci sont perdues. Si ces directives se perdent, c'est parce que SNMP est un protocole qui fonctionne sur UDP, donc en mode non-connecté. il n'y a donc pas de retour une fois la donnée envoyée (ACK en TCP). il faut donc contrôler que la configuration a bien été appliquée sur l'actif, et c'est l'API la mieux placée pour cela.

MISE EN OEUVRE DU PROJET

OSSATURE DU LOGICIEL

Conteneurisation

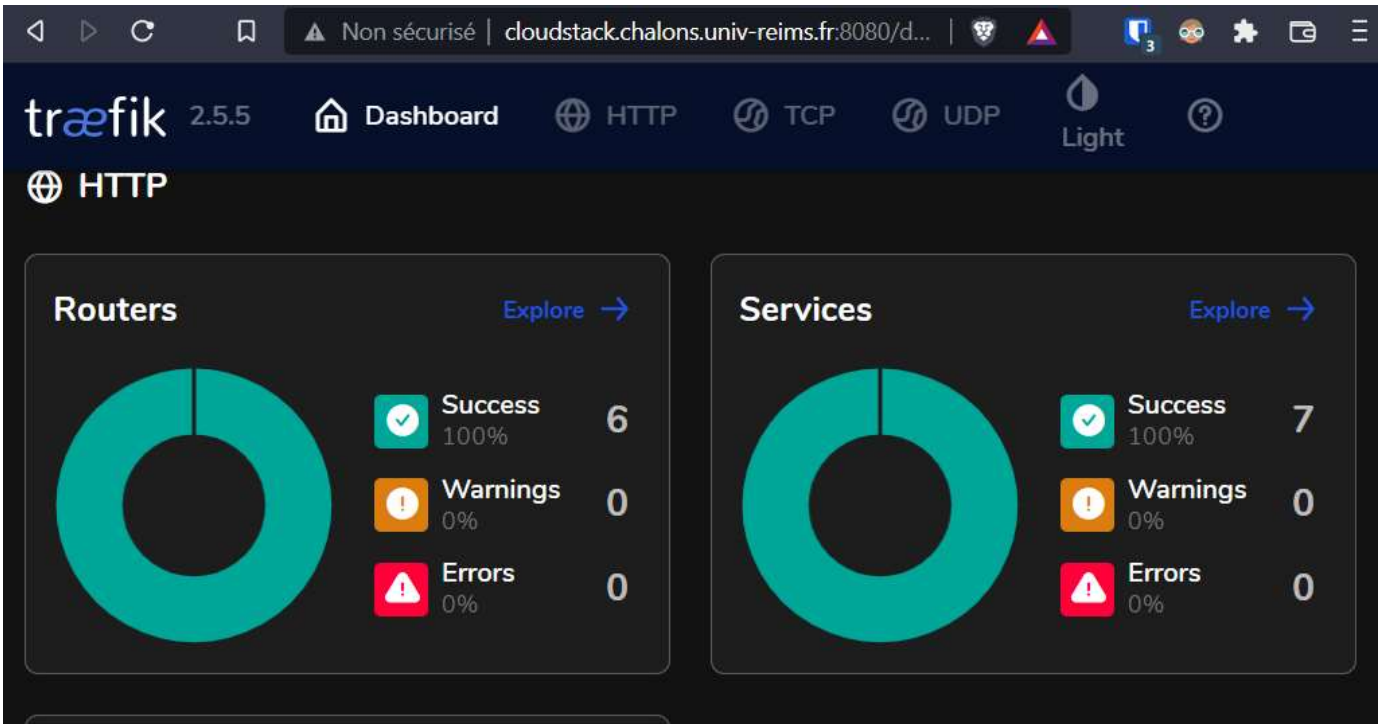
L'un des principes que nous avons tenu à mettre en place dans le cadre de notre projet est de conteneuriser nos applications. La **conteneurisation** est une technique qui consiste à encapsuler des applications dans des unités indépendantes et isolées. Cela permet un déploiement rapide, et plus de sécurité. Cela nous permet une plus grande **modularité**, puisque l'installation qui était autrefois fastidieuse, et qui impactait l'environnement du système, n'est plus nécessaire. Pour déployer une application, il suffit d'une seule commande, ou d'un bloc dans un fichier YAML. Le paquet que nous utilisons est **Docker**, qui est un service de conteneurisation assez répandu. Dans le cadre de notre projet, la conteneurisation a été mise en place, de façon à ce que tous les services nécessaires soient **instantiables** (puissent être démarrés) d'une seule commande. Voici les services qui sont en production, et qui sont tous démarrés en tant que **conteneurs** :

- **DNS** : Un service BIND qui tourne avec les fichiers de configuration en lecture seule.
- **Base de données** : Une base de données MongoDB, avec les fichiers de la base dans un répertoire /home/mongo.
- **Serveur web** : Un serveur web pour la documentation (sur laquelle vous êtes), avec les fichiers HML générés par Hugo en lecture seule.
- **APIs** : Les APIs de l'application, déployées pour les salles Réseaux 1, 2 et 3, ainsi que pour un actif de test.
- **Gestionnaire de topologie** : Le gestionnaire de topologie développé pour le projet.
- **Traefik** : Le reverse proxy, permet de fournir le bon service, en fonction de l'URL demandée.

Pour démarrer ces services d'une seule commande, nous utilisons **Docker Compose**. Docker Compose est un outil qui permet de déployer des conteneurs, et de les configurer dans un fichier YAML. Le fichier YAML de configuration utilisé pour notre projet est proposé en annexe 6.

Traefik

Traefik est un **reverse proxy**, dont la spécificité est de découvrir les services à fournir depuis le service Docker. Sa configuration se fait via des **tags** appliqués aux conteneurs, dans le fichier de Docker Compose. Via ces tags, nous pouvons configurer l'URL qui route sur le service, le port du service, et d'autres paramètres utiles pour un reverse proxy. Traefik fournit aussi une interface web, qui permet de surveiller le bon fonctionnement du service.



Évidemment, le service de reverse proxy ne fonctionnerait pas pour un utilisateur, sans résolution de nom, il fonctionne donc en union avec le DNS.

DNS et Délégation de zone

Le **DNS** permet de traduire des noms de domaine en adresses IP. Il est un des fondements d'Internet, et notre projet ne déroge pas à ce principe. Dans le cadre de notre projet, plusieurs ressources nous ont été accordées :

- un enregistrement A : cloudstack.chalons.univ-reims.fr
- une délégation de zone : sdn.chalons.univ-reims.fr

Ces ressources ont été attribuées comme suit :

Service	Adresse	Port
Interface de gestion Traefik	cloudstack.chalons.univ-reims.fr	8080
Documentation et rapport de projet	doc.sdn.chalons.univ-reims.fr	80
Gestionnaire de topologie et interface web	sdn.chalons.univ-reims.fr	80
APIs	R[123].api.sdn.chalons.univ-reims.fr	80

OUTILS DE COLLABORATION

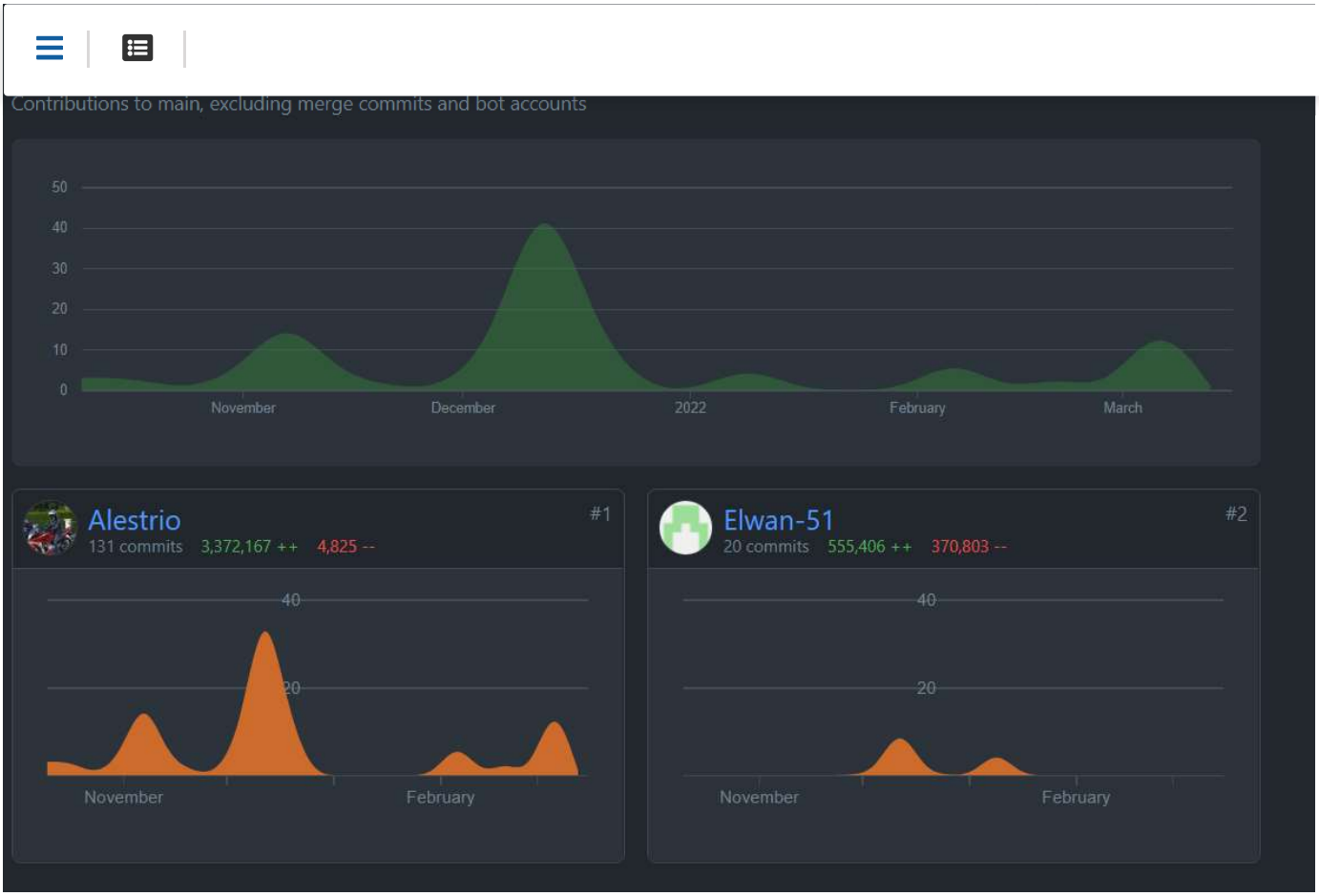
Cette section est consacrée aux outils de collaboration que nous avons utilisé pendant la réalisation de notre projet.

Hugo

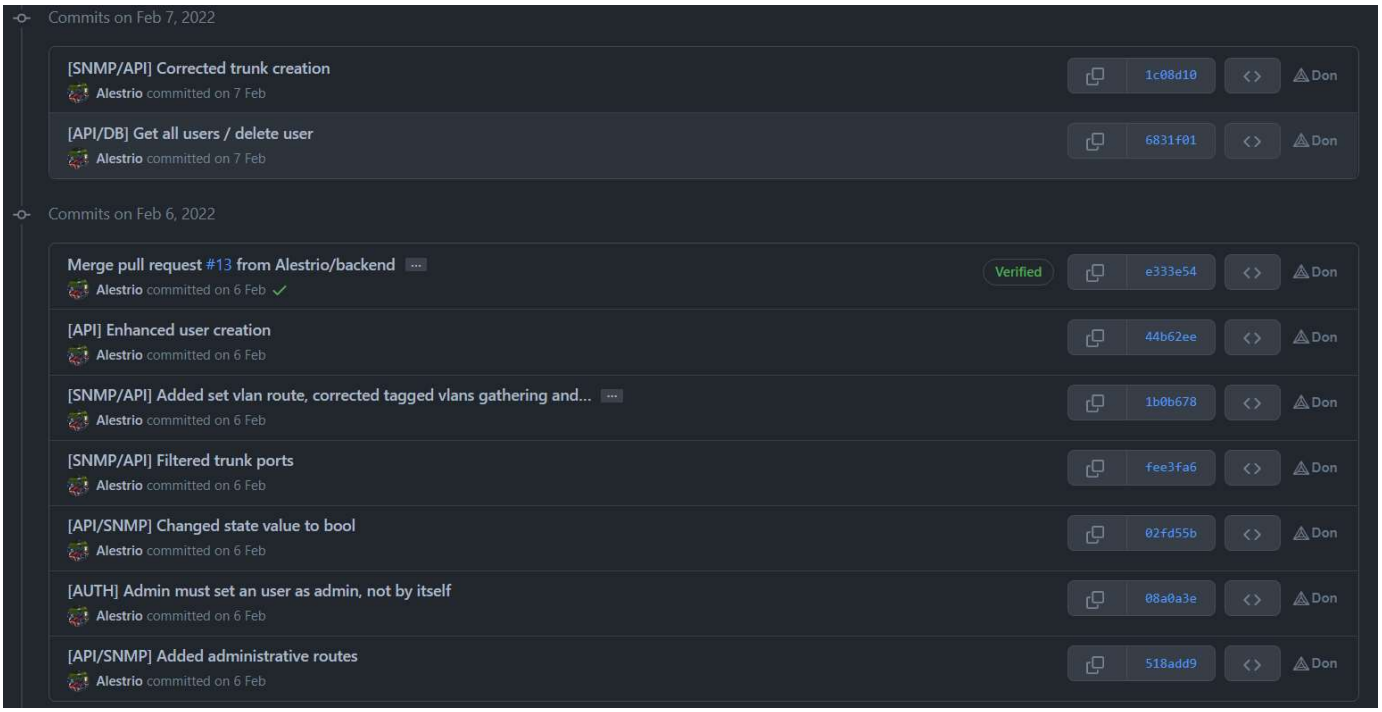
Hugo est un générateur de **site web statique**, à partir de fichiers **Markdown**. Nous l'avons utilisé tout au long de ce projet pour créer une documentation pour le projet. De cette façon, nous avions par écrit, toutes les informations techniques concernant le projet, ce qui nous a permis de ne pas devoir dépendre continuellement de l'autre membre de l'équipe afin de comprendre et utiliser le travail réalisé. C'est aussi l'outil que nous utilisons afin de créer ce rapport. L'ensemble des fichiers Markdown, sont hébergés sur GitHub, à cette adresse : https://github.com/Alestrio/SDN-Hugo

Github

Github est une plateforme collaborative reposant sur **Git**, un utilitaire de gestion de version. Cette plateforme nous a permis de travailler chacun de notre côté sur nos tâches respectives, puis de **fusionner** les sources des deux membres de l'équipe. Ainsi, nous évitons les **conflits** de version, et nous avons un **historique** complet de toutes les versions du projet.



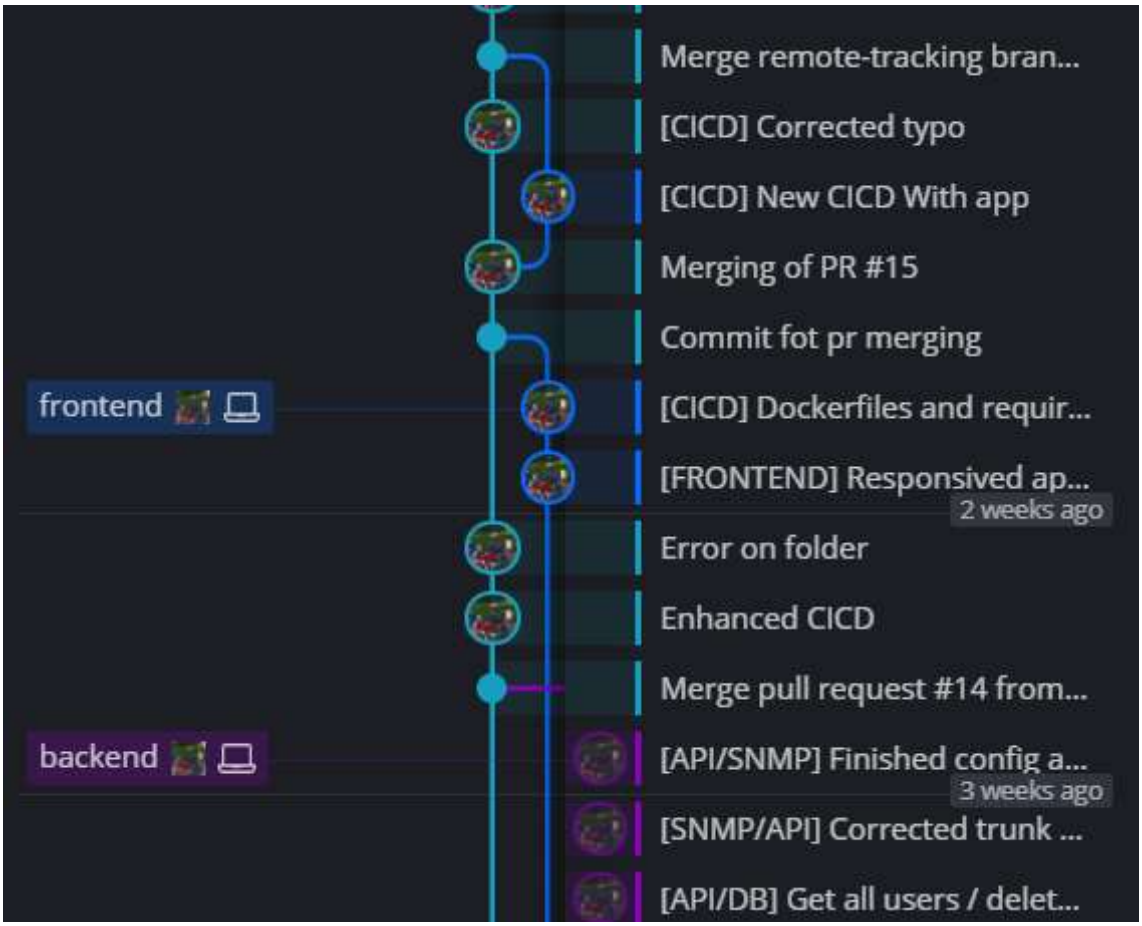
Contributions de l'équipe sur Github



Extrait d'historique du projet sur Github

Afin de travailler chacun de notre côté sur nos tâches respectives, nous avons utilisé des **branches**. Une branche est l'équivalent d'un carrefour où une route se divise en plusieurs chemins.

Nous avons créé une branche backend, pour travailler sur tout ce qui concerne les APIs, puis une branche frontend, pour travailler sur tout ce qui concerne le gestionnaire de topologies et l'interface web. L'ensemble est ensuite intégré au projet en production, via la création d'une **Pull request**, qui permet de faire une demande de **fusion** entre deux branches. La branche **master** est ainsi la branche principale, et c'est celle qui est utilisée lors du déploiement automatique du projet.

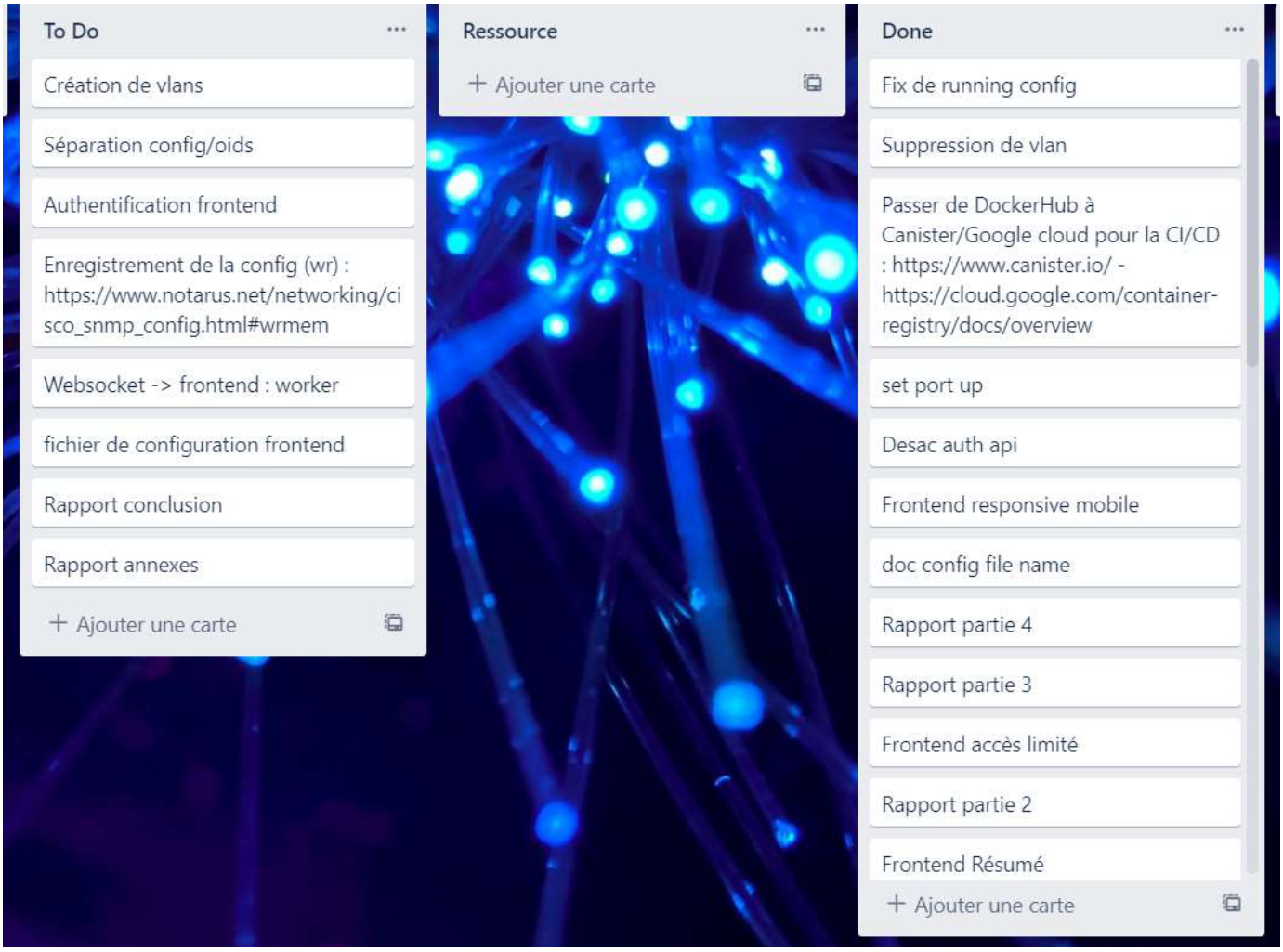


Branches de travail sur Github

Dans un contexte de développement collaboratif avec une **production**, un service crucial qui doit être accessible constamment, la manière de s'organiser n'est pas tout à fait la même. Généralement, il y a trois branches principales : dev, preprod et prod. Dans le cas de Cloudstack, nous avons décidé de ne pas suivre cette architecture, puisque celle-ci est

Trello

Afin de garder une trace sur les tâches faites/à faire, nous nous sommes aidés d'un outil appellé **Trello**. Cet outil est un outil de gestion de projet, qui permet de gérer les tâches, les priorités, les échéances, les tags, les membres, etc. Nous avons utilisé Trello pour gérer les tâches, et plusieurs colonnes : A faire, En cours, Fait, A regarder. Au fur et à mesure que nous avons travaillé sur nos tâches, nous avons ajouté des tâches dans les colonnes correspondantes, puis nous les avons fait passer dans les colonnes suivantes : En cours, Fait. Cette façon d'organiser les choses s'appelle le **Kanban**.



Extrait de notre Kanban

Afin de ne pas avoir à redéployer le projet manuellement à chaque modification, nous avons mis en place une **CI/CD**. CI/CD signifie **Continuous Integration/Continuous Deployment**. C'est une automatisation qui permet de lancer des tests à chaque modification sur un projet, mais aussi de le redéployer en production lors d'une modification sur une branche spéciale. Dans notre cas, c'est la branche **master**, qui est utilisée pour le déploiement automatique.

La machine virtuelle qui est utilisée pour le déploiement automatique est un serveur Ubuntu, qui est hébergé sur le datacenter de l'IUT. Sur cette machine, nous avons installé un **worker**, qui est un service qui permet de lancer les actions que nous avons définies dans le fichier .github/workflows/deploy_dc_chalons.yml. Ce fichier définit les actions à effectuer lors d'un déploiement automatique. Dans notre cas, voici les actions effectuées :

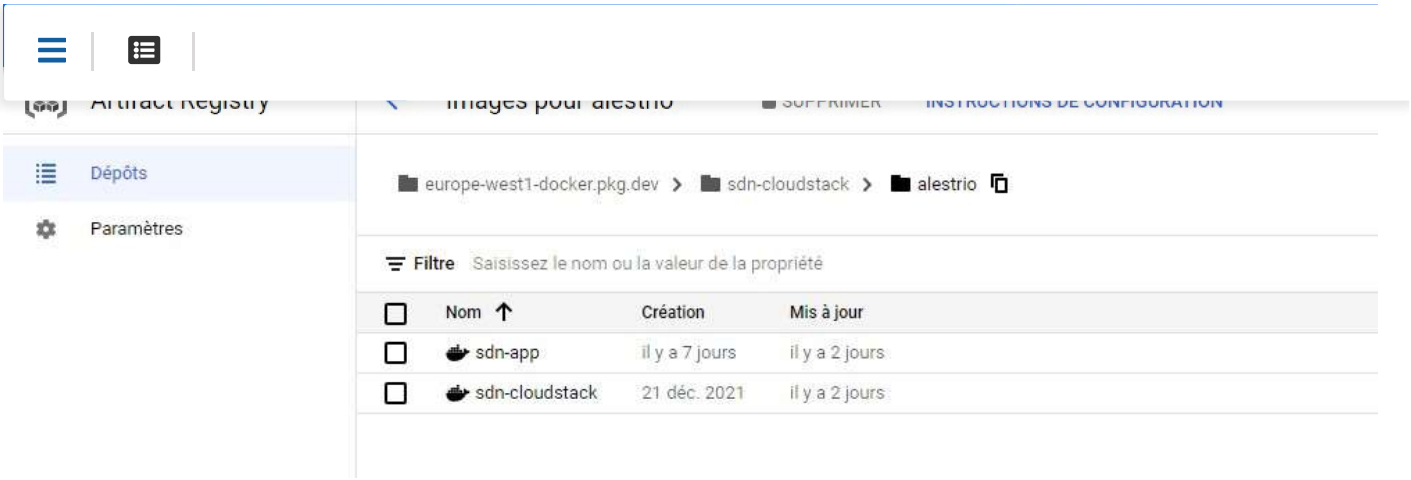
- **Pull** : permet de récupérer les modifications depuis le serveur de GitHub sur le serveur de déploiement.
- **Build** : permet de créer les images Docker pour le déploiement automatique.
- **Upload** : permet d'envoyer les images Docker sur l'hébergeur d'images (ici, Google Cloud, Artifact Registry).
- **Restart** : redémarre le Docker-Compose en production.

Une fois ces actions effectuées, la version en production est mise à jour.

Ce fichier est proposé en annexe 7.

Google Cloud, Artifact Registry

Afin de stocker les images Docker, pour les récupérer sur n'importe quelle machine, nous utilisons **Google Artifact Registry**. C'est un service qui permet de stocker des images Docker. Il suffit ensuite de précisez l'adresse de l'image Docker pour la déployer.



Google Cloud, Artifact Registry

```
app:
  restart: always
  image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-app:latest
```

Déploiement sur serveur

EVALUATION

Cette section est consacrée à notre évaluation objective et subjective du projet.

Evaluation objective

Par rapport au cahier des charges

Durant ce projet, nous avons pu développer un très grande majorité (\approx 80%) de ce qui était demandé dans le **cahier des charges**. Nous avons donc créé un système de configuration des actifs par **SNMP**.
Nous avons aussi développé une **interface utilisateur** avec gestion des configurations préenregistrées, configuration à la volée et affichage des informations sur les actifs.
Cette interface a été développée en gardant en tête que le principal objectif était de pouvoir utiliser ce système sur **mobile**.

Hors du cahier des charges

Nous avons pu déployer notre projet en utilisant une infrastructure conteneurisée, à l'aide de **Docker** et Docker-Compose, ce qui nous a permis de nous rapprocher au maximum des nouvelles technologies de développement.
Nous avons également utilisé un système de gestion de version **Git**, ce qui nous a permis de suivre les changements du code, et de mettre en place un système de déploiement automatique, aussi appellé CI/CD.
Avec ce projet, nous avons aussi pu mettre en place une infrastructure de production complète, avec gestion de zone **DNS**, proxy **Traefik**, serveur web, serveur de bases de données **MongoDB**, etc.

Des tâches qui restent à faire

Nous avons néanmoins pu réfléchir au principe de base de cette fonctionnalité, et nous avions une idée assez précise de la mise en place de cette fonctionnalité.
Il reste aussi du travail au niveau de l'interface web, dont on pourrait améliorer l'esthétique et la facilité d'utilisation.

Evaluation subjective

Compétences acquises

Grâce à ce projet, nous avons pu acquérir des compétences en système, à travers l'utilisation de technologies comme **Docker, Docker-Compose, SNMP, Traefik**, etc.
Nous avons aussi pu acquérir des compétences en développement, à travers l'utilisation de technologies comme **Git, Github, et la CI/CD.**
Enfin, nous avons pu acquérir des compétences en développement web, à travers l'utilisation de technologies comme **FastAPI, Flask,** etc.

Des choix divergents par rapport au cahier des charges

Nous avons fait **deux choix majeurs** par rapport au cahier des charges.
Tout d'abord, nous avions à la base choisi de gérer l'**authentification** et l'autorisation des utilisateurs, depuis l'**API**, qui aurait été le **centre nevralgique** de notre application. Ce choix s'est avéré trop **complexe** par rapport au cahier des charges, et nous a fait perdre un peu de temps pendant le développement de l'API et de l'interface web.
Nous avons donc choisi de gérer l'authentification et l'autorisation des utilisateurs, depuis l'interface web, avec les identifiants et les mots de passes stockés dans un fichier de configuration.
Le deuxième et dernier choix que nous avons fait, est de gérer les **interactions** avec la base de données et les actifs, uniquement à travers l'API.
Ce choix a été fait pour nous rapprocher au maximum des techniques aujourd'hui utilisées en **production** dans une grande majorité d'applications web.

CONCLUSION

Pour conclure ce rapport de projet, on peut dire que le projet a été réalisé en garned partie avec succès. Nous avons pu développer un interface de configuration des actifs d'ancienne génération, ainsi que l'interface web pour utiliser cette interface de configuration.
Pendant ce projet, nous avons pu utiliser beaucoup de technologies aujourd'hui utilisées en production, pour des applications de grande envergure.

Personnellement, nous avons beaucoup appris grâce à ce projet. Nous avons pu apprendre à nous servir des nouvelles technologies de production (Conteneurisation, Traefik, Déploiement automatique, etc.)
Nous avons aussi pu consolider nos connaissances en programmation, à travers le développement de l'interface web et de l'interface de configuration (avec des technologies comme FastAPI, Flask, etc.).
Enfin grâce à ce projet, nous avons pu consolider nos compétences de travail en équipe.

Nous espérons pouvoir nous servir à nouveau de ces connaissances pour des projet futurs !

INDEX

ACCESS

Le mode access est un mode de configuration de port sur les équipements Cisco qui est utilisé pour la connexion de périphérique (ordinateur, imprimante ...).

API

API signifie **Application Programming Interface**, une API est dans les faits un service qui expose des procédures de configuration et de récupération de données, par des URL paramétrables.

BACKBONE

Le backbone est-ce que l'on considère comme la colonne vertébrale d'un projet ou d'une application.

CI/CD

CI/CD signifie **intégration continue / déploiement continu**. C'est un procédé qui permet de publier automatiquement sur un serveur les avancées d'un projet quand on le juge acceptable. Dans notre cas nous l'utilisons pour publier les mises à jour de l'application ainsi que pour publier notre rapport de projet.

DOCKER

Docker est un logiciel qui permet de construire, exécuter et manager des "conteneur" sur des serveurs ou directement dans le cloud. Les conteneurs quant à eux sont des fichiers ou ensembles de fichier qui définissent une application pouvant être exécutée par le service Docker. Concrètement, un conteneur est un environnment d'exécution isolé pour une application.

GIT

Git est un système de gestion de version. C'est un système qui permet le travail collaboratif, il permet de travailler sur des fichiers et de rendre le tout commun en évitant les problèmes de version et de conflits.

GITHUB

Github est une application en ligne, basé sur git, elle permet le management de projet, mais, également le partage d'idée pour les projets publics. Nous utilisons dans notre cas Github pour héberger les fichiers du projet et également déployer ceux-ci sur les serveurs de l'IUT avec la CI/CD.

JSON

JSON signifie **JavaScript Object Notation**, c'est un format de fichier très utilisé dans le monde du développement, particulièrement pour l'écriture de fichiers de configuration.

PARC D'ACTIFS

Le parc d'actifs d'un réseau est l'ensemble des équipements réseaux qui sont en production sur un réseau.

PORT CONSOLE

Le port console est un port physique dédié à la configuration d'un actif réseaux.

SNMP

SNMP signifie **Simple Network Management Protocol**. C'est un protocole qui permet à différent équipements sur un réseau de partager des informations. Il est particulièrement utile, car, il permet à des équipements de natures différentes (switch et routeur par exemple) et de versions différentes de communiquer.

SSH

SSH signifie **Secure SHell**. SSH est un services de communication sécurisé, cela signifie que les informations transmises au travers du réseau sont chiffrées.

TCP

TCP signifie **Transmission Control Protocol**. TCP est un protocole de transport des informations standard sur un réseau. Il est utilisé afin d'être sûr que l'intégralité des données sont transmises lors d'une communication.

TELNET

TELNET signifie **TELecommunication NETwork**. TELNET est un protocole similaire à SSH cependant celui-ci n'est pas sécurisé laissant apparaitre les informations transmises librement sur le réseau.

TRUNK

TRUNK est l'un des deux modes que peuvent prendre les ports d'un actif réseau. Il est utilisé dans le cas ou plusieurs vlans doivent circuler au même endroittravers d'un même port.

UDP

UDP signifie **User Datagram Protocol**. UDP est un mode de transport similaire à TCP, cependant celui-ci ne s'assure pas de la bonne réception des données transmises.

VLAN

VLAN signifie **Virtual LAN**. C'est un sous-réseau virtuel qui permet de regrouper différents équipements à l'intérieur d'une "unité de commutation", c'est l'équivalent d'une boite noire, elle permet de regrouper les équipements sur un même réseau.

YAML

YAML est un langage permettent d'ordonner des données similaires à JSON. Il est souvent utilisé pour les fichiers de configuration.

1. EXEMPLE DE TEMPLATE FLASK

```
<header class=" header flex w-full h-14 md:h-20  items-center text-2xl font-extrabold text-white space-x-5">
    <div>
        <button class="px-5 grow hidden md:block"><a href="/">SDN-Cloudstack</a></button>
    </div>
    <!---- Menu fold button ---->
    <div class="flex-1 flex justify-end items-center align-middle">
        <button class="block md:hidden" onclick="toggleMenu()">
            <svg class="h-6 w-6 fill-current text-white" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">
                <path d="M0 3h20v2H0V3zm0 6h20v2H0V9zm0 6h20v2H0v-2z"/>
            </svg>
        </button>
    <div>
        <select class="header text-white text-sm md:text-xl align-middle" onchange="if (this.value) window.location.href=this.value">
            {%for room in api%}
                {% if selected == room %}
                    <option value="./{{room}}" selected>{{room}}</option>
                {%else%}
                    <option value="./{{room}}">{{room}}</option>
                {%endif%}
            {%endfor%}
        </select>
    </div>
    <div class="flex-auto text-center">
        <p class="text-sm md:text-xl align-middle">{{title}}</p>
    </div>
    {% if user %}
        <div class="flex-auto text-center">
            <button class="px-5 text-sm md:text-xl align-middle"><a href="/logout">{{ user }}</a></button>
        </div>
    {%else%}
        <div class="flex-auto text-center">
```

```
            <button class="px-5 text-sm md:text-xl align-middle"><a
href="/login">Login</a></button>
        </div>
    {%endif%}
</header>
```

Les blocs entre crochets sont des blocs de pseudo-python intégrés dans le template.

2. CODE MISES À JOUR AUTOMATIQUE

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('', 514))

async def listen(websocket, path):
    print("Start listening..")
    REGEX = # regex pour conertir le log en évenement
    while True:
        print('listening...')
        data = s.recv(4048)
        data = data.decode('utf-8')
        print(data)
        matches = re.search(REGEX, data)
        if matches:
            ...
            await websocket.broadcast(json.dumps(onu_info))
            #server.send_message_to_all(json.dumps(onu_info))
        else:
            await websocket.broadcast(json.dumps({"message": "No matches
found"}))


async def main():
    print("Running.")
    async with websockets.serve("0.0.0.0", 6969):
        await asyncio.Future()  # run forever

asyncio.run(main())
asyncio.run(listen())
```

Code réalisé en collaboration avec le groupe de projet AC-Vision.

3. LISTE DES OIDS

```yaml
oids:
  systemName: 1.3.6.1.2.1.1.5
  uptime: 1.3.6.1.2.1.1.3
  cdp_neighbors:
    ip: 1.3.6.1.4.1.9.9.23.1.2.1.1.4
    fqdn: 1.3.6.1.4.1.9.9.23.1.2.1.1.6
    interface: 1.3.6.1.4.1.9.9.23.1.2.1.1.7
    model: 1.3.6.1.4.1.9.9.23.1.2.1.1.8
  interfaces:
    table: 1.3.6.1.2.1.2.2
    oids:
      description: 1.3.6.1.2.1.2.2.1.2
      index: 1.3.6.1.2.1.2.2.1.1
      status: 1.3.6.1.2.1.2.2.1.7
      speed: 1.3.6.1.2.1.2.2.1.5
      vlan: 1.3.6.1.4.1.9.9.68.1.2.2.1.2
      operstatus: 1.3.6.1.2.1.2.2.1.8
      mac_address: 1.3.6.1.2.1.2.2.1.6
      sets:
        vlan: 1.3.6.1.4.1.9.9.68.1.2.2.1.2
        trunk_mode: 1.3.6.1.4.1.9.9.46.1.6.1.1.13
        access: 1.3.6.1.4.1.9.9.46.1.6.1.1.29 # 1-trunk,2-access,3-disabled
    mib_names:
      description: ifDescr
      index: ifIndex
      status: ifAdminStatus
      speed: ifSpeed
      operstatus: ifOperStatus
      mac_address: ifPhysAddress
  trunks:
    table: 1.3.6.1.4.1.9.9.46.1.6.1
    oids:
      index: 1.3.6.1.4.1.9.9.46.1.6.1.1.1
      domain: 1.3.6.1.4.1.9.9.46.1.6.1.1.2
      vlans: 1.3.6.1.4.1.9.9.46.1.6.1.1.4
      vlans2k: 1.3.6.1.4.1.9.9.46.1.6.1.1.17
      vlans3k: 1.3.6.1.4.1.9.9.46.1.6.1.1.18
      vlans4k: 1.3.6.1.4.1.9.9.46.1.6.1.1.19
      native: 1.3.6.1.4.1.9.9.46.1.6.1.1.5
      status: 1.3.6.1.4.1.9.9.46.1.6.1.1.13
```

```
    encapsulation: 1.3.6.1.4.1.9.9.46.1.6.1.1.3
vlans:
  table: 1.3.6.1.4.1.9.9.46.1.3.1
  name: 1.3.6.1.4.1.9.9.46.1.3.1.1.4.1
  dot1q_id: 1.3.6.1.4.1.9.9.46.1.3.1.1.6.1
  set:
    operation: 1.3.6.1.4.1.9.9.46.1.4.1.1.1
    row_status: 1.3.6.1.4.1.9.9.46.1.4.2.1.11.1
    type: 1.3.6.1.4.1.9.9.46.1.4.2.1.3.1
    name : 1.3.6.1.4.1.9.9.46.1.4.2.1.4.1
    said: 1.3.6.1.4.1.9.9.46.1.4.2.1.6.1
```

4. URLS DE L'API

FastAPI 0.1.0 OAS3

/openapi.json

Authorize

General Infos

GET	/api/v1.5/neighbors	Get Cdp Neighbors
GET	/api/v1.5/hostname	Get Hostname
GET	/api/v1.5/uptime	Get Uptime

Interfaces

GET	/api/v1.5/interfaces	Get Interfaces
GET	/api/v1.5/interfaces/{if_id}	Get If Id
GET	/api/v1.5/interfaces/by_vlan/{vlan_id}	Get Interfaces By Vlan
POST	/api/v1.5/interfaces/{if_id}/vlan/{vlan_id}	Set Vlan On Interface
POST	/api/v1.5/interface/{if_id}/state/{state}	Set Interface State
POST	/api/v1.5/interface/{if_id}/description	Set Interface Description

VLANs

GET	/api/v1.5/vlans	Get Vlans
POST	/api/v1.5/vlans	Create Vlan

GET	/api/v1.5/vlans/{vl_id}	Get Vlan Id

Trunks

GET	/api/v1.5/trunks	Get Trunks
GET	/api/v1.5/trunks/brief	Get Trunks Brief
GET	/api/v1.5/trunks/{id}	Get Trunk
GET	/api/v1.5/trunks/dot1q_id/{dot1q_id}	Get Trunk By Dot1Q Id
POST	/api/v1.5/trunks/set	Set Trunk
POST	/api/v1.5/trunks/{tr_id}/tagged_vlan/{vlan_id}	Tagged Vlan
PUT	/api/v1.5/trunks/{tr_id}/native_vlan/{vlan_id}	Native Vlan

Config

POST	/api/v1.5/config	Add Config
GET	/api/v1.5/configs	Get Configs
GET	/api/v1.5/configs/running	Get Running Config
GET	/api/v1.5/configs/running_cisco	Get Running Config Cisco
GET	/api/v1.5/configs/{config_id}	Get Config
DELETE	/api/v1.5/configs/{config_id}	Delete Config
GET	/api/v1.5/configs/brief	Get Brief Configs
POST	/api/v1.5/configs/apply/{config_id}	Apply Config

Miscellaneous

GET `/api/v1.5/rebuild` Rebuild Cache

Schemas

Config

HTTPValidationError

Interface

Trunk

TrunkIn

ValidationError

Vlan

5. EXEMPLE DE TOPOLOGIE


```
{
  "name": null,
  "hostname": "\"RT-RES1.chalons.univ-reims.fr\"",
  "interfaces": [
    {
      "name": "Vlan1",
      "description": "Vlan1",
      "port_id": 1,
      "status": "up",
      "operstatus": "up",
      "vlan": null,
      "speed": 1000000000
    },
    {
      "name": "Vlan10",
      "description": "Vlan10",
      "port_id": 10,
      "status": "up",
      "operstatus": "up",
      "vlan": null,
      "speed": 1000000000
    },
    {
      "name": "StackPort1",
      "description": "StackPort1",
      "port_id": 5179,
      "status": "up",
      "operstatus": "up",
      "vlan": null,
      "speed": 0
    },
    {
      "name": "StackSub-St1-1",
      "description": "StackSub-St1-1",
      "port_id": 5180,
      "status": "up",
      "operstatus": "up",
      "vlan": null,
      "speed": 0
    },
```

```
        {
          "name": "StackSub-St1-2",
          "description": "StackSub-St1-2",
          "port_id": 5181,
          "status": "up",
          "operstatus": "up",
          "vlan": null,
          "speed": 0
        },
        {
          "name": "StackPort2",
          "description": "StackPort2",
          "port_id": 5182,
          "status": "up",
          "operstatus": "up",
          "vlan": null,
          "speed": 0
        },
        {
          "name": "StackSub-St2-1",
          "description": "StackSub-St2-1",
          "port_id": 5183,
          "status": "up",
          "operstatus": "up",
          "vlan": null,
          "speed": 0
        },
        {
          "name": "StackSub-St2-2",
          "description": "StackSub-St2-2",
          "port_id": 5184,
          "status": "up",
          "operstatus": "up",
          "vlan": null,
          "speed": 0
        },
        {
          "name": "FastEthernet1/0/1",
          "description": "FastEthernet1/0/1",
          "port_id": 10001,
          "status": "up",
          "operstatus": "down",
          "vlan": {
            "description": "SALLE-RT1-PORT-1",
            "dot1q_id": 201
          },
          "speed": 10000000
        },
        {
          "name": "FastEthernet1/0/2",
          "description": "FastEthernet1/0/2",
          "port_id": 10002,
          "status": "up",
```

```
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-2",
        "dot1q_id": 202
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/3",
      "description": "FastEthernet1/0/3",
      "port_id": 10003,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-3",
        "dot1q_id": 203
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/4",
      "description": "FastEthernet1/0/4",
      "port_id": 10004,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-4",
        "dot1q_id": 204
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/5",
      "description": "FastEthernet1/0/5",
      "port_id": 10005,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-5",
        "dot1q_id": 205
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/6",
      "description": "FastEthernet1/0/6",
      "port_id": 10006,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-6",
        "dot1q_id": 206
      },
```

```
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/7",
      "description": "FastEthernet1/0/7",
      "port_id": 10007,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-7",
        "dot1q_id": 207
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/8",
      "description": "FastEthernet1/0/8",
      "port_id": 10008,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-8",
        "dot1q_id": 208
      },
      "speed": 10000000
    },
    {
      "name": "FastEthernet1/0/9",
      "description": "FastEthernet1/0/9",
      "port_id": 10009,
      "status": "up",
      "operstatus": "down",
      "vlan": {
        "description": "SALLE-RT1-PORT-9",
        "dot1q_id": 209
      },
      "speed": 10000000
    },
   [...]
  ],
  "vlans": [
    {
      "description": "rt-reseau-2",
      "dot1q_id": 68
    },
    {
      "description": "rt-reseau-3",
      "dot1q_id": 72
    },
    {
      "description": "RT1G1B1-0",
      "dot1q_id": 300
    },
```

```
      {
        "description": "RT1G1B1-1",
        "dot1q_id": 301
      },
      {
        "description": "RT1G1B1-2",
        "dot1q_id": 302
      },
      {
        "description": "RT1G1B2-0",
        "dot1q_id": 303
      },
      {
        "description": "RT1G1B2-1",
        "dot1q_id": 304
      },
      {
        "description": "RT1G1B2-2",
        "dot1q_id": 305
      },
      {
        "description": "RT1G1B3-0",
        "dot1q_id": 306
      },
     [...]
    ],
    "trunks": [
      {
        "interface": {
          "name": "FastEthernet2/0/32",
          "description": "FastEthernet2/0/32",
          "port_id": 10532,
          "status": "up",
          "operstatus": "down",
          "vlan": null,
          "speed": 100000000
        },
        "native_vlan": {
          "description": "RT2G1B1-0",
          "dot1q_id": 400
        },
        "tagged_vlans": [
          0,
          4,
          8,
          16,
          26,
          336,
          337,
          338
        ],
        "status": "1"
      },
```

```
{
  "interface": {
    "name": "FastEthernet2/0/33",
    "description": "FastEthernet2/0/33",
    "port_id": 10533,
    "status": "up",
    "operstatus": "down",
    "vlan": null,
    "speed": 100000000
  },
  "native_vlan": {
    "description": "RT2G1B2-0",
    "dot1q_id": 403
  },
  "tagged_vlans": [
    0,
    4,
    8,
    16,
    26,
    339,
    340,
    341
  ],
  "status": "1"
},
{
  "interface": {
    "name": "FastEthernet2/0/34",
    "description": "FastEthernet2/0/34",
    "port_id": 10534,
    "status": "up",
    "operstatus": "down",
    "vlan": null,
    "speed": 100000000
  },
  "native_vlan": {
    "description": "RT2G1B3-0",
    "dot1q_id": 406
  },
  "tagged_vlans": [
    0,
    4,
    8,
    16,
    26,
    342,
    343,
    344
  ],
  "status": "1"
},
{
```

```
      "interface": {
        "name": "FastEthernet2/0/35",
        "description": "FastEthernet2/0/35",
        "port_id": 10535,
        "status": "up",
        "operstatus": "down",
        "vlan": null,
        "speed": 100000000
      },
      "native_vlan": {
        "description": "RT2G1B4-0",
        "dot1q_id": 409
      },
      "tagged_vlans": [
        0,
        4,
        8,
        16,
        26,
        345,
        346,
        347
      ],
      "status": "1"
    },
    [...]
  ]
}
```

6. FICHIER DOCKER-COMPOSE

```
version: "3"


services:
  hugo:
    image: nginx:latest
    restart: always
    volumes:
      - /home/user/compose/hugo/sdn_notice/public/:/usr/share/nginx/html/
    labels:
        - "traefik.enable=true"
        - "traefik.http.routers.doc.rule=Host(`doc.sdn.chalons.univ-
reims.fr`)"
    networks:
        backend:
  reverse-proxy:
    # The official v2.0 Traefik docker image
    image: traefik:latest
    # Enables the web UI and tells Traefik to listen to docker
    restart: always
    command:
      --api.insecure=true
      --providers.docker
      --entrypoints.web.address=:80
    ports:
      # The HTTP port
      - "80:80"
      # The Web UI (enabled by --api.insecure=true)
      - "8080:8080"
    volumes:
      # So that Traefik can listen to the Docker events
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/user/compose/traefik/traefik.toml:/etc/traefik/traefik.toml
      - /home/user/compose/traefik/acme.json:/acme.json
      - /home/user/compose/traefik/access.log:/var/log/access.log
```

```
    networks:
        backend:
  api_r1:
    restart: always
    image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-
cloudstack:latest
    volumes:
      - /home/user/compose/api/r1/config.yaml:/home/api/config/config.yaml:ro
    environment:
      - CONFIG_DIR=/home/api/config/
    labels:
      - "traefik.enable=true"
      - "traefik.http.services.api_r1.loadbalancer.server.port=8000"
      - "traefik.http.routers.api_r1.rule=Host(`r1.api.sdn.chalons.univ-
reims.fr`)"
    networks:
        backend:
  api_r2:
    restart: always
    image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-
cloudstack:latest
    volumes:
      - /home/user/compose/api/r2/config.yaml:/home/api/config/config.yaml:ro
    environment:
      - CONFIG_DIR=/home/api/config/
    labels:
      - "traefik.enable=true"
      - "traefik.http.services.api_r2.loadbalancer.server.port=8000"
      - "traefik.http.routers.api_r2.rule=Host(`r2.api.sdn.chalons.univ-
reims.fr`)"
    networks:
        backend:
  api_r3:
    restart: always
    image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-
cloudstack:latest
    volumes:
      - /home/user/compose/api/r3/config.yaml:/home/api/config/config.yaml:ro
    environment:
      - CONFIG_DIR=/home/api/config/
    labels:
      - "traefik.enable=true"
      - "traefik.http.services.api_r3.loadbalancer.server.port=8000"
      - "traefik.http.routers.api_r3.rule=Host(`r3.api.sdn.chalons.univ-
reims.fr`)"
    networks:
        backend:
  api_test:
    restart: always
    image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-
cloudstack:latest
    volumes:
      -
```

```
/home/user/compose/api/test/config.yaml:/home/api/config/config.yaml:ro
    environment:
      - CONFIG_DIR=/home/api/config/
    labels:
      - "traefik.enable=true"
      - "traefik.http.services.api_test.loadbalancer.server.port=8000"
      - "traefik.http.routers.api_test.rule=Host(`test.api.sdn.chalons.univ-
reims.fr`)"
    networks:
        backend:
  app:
    restart: always
    image: europe-west1-docker.pkg.dev/sdn-cloudstack/alestrio/sdn-app:latest
    labels:
      - "traefik.enable=true"
      - "traefik.http.services.app.loadbalancer.server.port=5000"
      - "traefik.http.routers.app.rule=Host(`www.sdn.chalons.univ-reims.fr`)"
    networks:
        backend:
  bind:
    restart: always
    image: sameersbn/bind:latest
    ports:
      - "53:53/udp"
      - "53:53/tcp"
      - "10000:10000/tcp"
    volumes:
      - /home/user/compose/bind/:/data
networks:
  backend:
```

7. FICHIER DE CICD

```yaml
name: Docker Image CI

on:
  push:
    branches: [ main ]

jobs:
  build:
    runs-on: sdn-api
    steps:
    - uses: actions/checkout@v2
      with:
        fetch-depth: 0
    - name: Build the Docker image for api
      run: docker image build . --file Dockerfile --tag europe-west1-
docker.pkg.dev/sdn-cloudstack/alestrio/sdn-cloudstack:latest
      working-directory: /home/user/SDN-Cloudstack/SDN-Cloudstack
    - name: Build the Docker image for app
      run: docker image build . --file Dockerfile --tag europe-west1-
docker.pkg.dev/sdn-cloudstack/alestrio/sdn-app:latest
      working-directory: /home/user/SDN-Cloudstack/SDN-Cloudstack/src/app
    - name: Docker push
      run: docker image push europe-west1-docker.pkg.dev/sdn-
cloudstack/alestrio/sdn-cloudstack:latest
      working-directory: /home/user/SDN-Cloudstack/SDN-Cloudstack
    - name: Docker push
      run: docker image push europe-west1-docker.pkg.dev/sdn-
cloudstack/alestrio/sdn-app:latest
      working-directory: /home/user/SDN-Cloudstack/SDN-Cloudstack/src/app
    - name: Docker stop
      run: docker-compose down
      working-directory: /home/user/compose/compose
    - name: Docker reup
      run: docker-compose up -d
      working-directory: /home/user/compose/compose
```