Automatiser son infrastructure réseau

Jérôme Durand
Cisco Systems
11 rue Camille Desmoulins
92130 Issy-les-Moulineaux

Résumé

L'automatisation des réseaux est un sujet qui est toujours plus au cœur des préoccupations des organisations. La situation actuelle où la configuration est faite manuellement, équipement par équipement, n'est plus tenable car le nombre d'équipements réseau et leur criticité ont augmenté beaucoup plus vite que les personnels en charge de leur management. Des solutions diverses existent pour répondre aux enjeux de l'automatisation réseau, mais leur mise en place doit être soigneusement réfléchie. Au delà d'outils, c'est toute l'architecture qui est concernée. Un outil qui devra générer des configurations spécifiques, équipement par équipement, ne sera pas efficace, alors qu'une refonte de l'architecture avec du contrôle d'accès et en mode fabric permettra peut-être rapidement d'automatiser et de sécuriser tous les ports d'accès au réseau. De la même manière peut-être découvrirez-vous après deux ans de labeur et de personnalisation de playbooks Ansible, que vous aurez finalement la même chose qu'une solution fournie clé en main par un éditeur logiciel, témoignant d'un manque d'étude en amont. Enfin la mise en place d'une automatisation réseau ne peut pas se faire sans un minimum d'outillage permettant de tester les changements afin de permettre de répondre à la première demande : avoir un réseau fonctionnel ! Bien évidemment, il est impossible de tout faire en un jour et le voyage vers l'automatisation doit être décomposé en de multiples étapes avec à chaque fois un gain opérationnel clair, mais il convient de ne pas perdre de vue une cible globale.

Il faut également rappeler que l'automatisation n'est qu'une des composantes de l'administration des réseaux informatiques. Les préoccupations des équipes opérationnelles portent autant également sur la sécurité des infrastructures, et la capacité à mieux détecter les problèmes et à y remédier. Automatiser un réseau n'a de sens que si l'on ne travaille pas en parallèle sur ces deux axes tout autant critiques.

Mots clé

802.1X, Ansible, Automation, CI/CD, Fabric, Jenkins, Netconf, Playbook, Radius, Restconf, VXLAN, YANG

1 Introduction

Les statistiques sont effrayantes : 95% des actions réalisées sur les réseaux sont faites manuellement. Si l'on met de côté les coûts mirobolants associés à ces opérations, cela a également des impacts très forts sur la sécurité des réseaux puisque près des trois quarts des incidents de sécurité sont associés à des erreurs de configuration. A l'heure où les applicatifs sont nettement automatisés, le monde du réseau peut sembler être resté à la préhistoire. Et pourtant, de nombreuses solutions permettent de sauter le pas et de programmer son réseau comme n'importe quelle application.

Si les constructeurs proposent des solutions clé-en-main, il est également possible de bâtir son propre environnement d'automatisation complètement personnalisé et adapté à l'infrastructure en place. Dans ce cas le risque principal est de se perdre dans des détails techniques loin des problématiques métier réelles.

Quelles topologies sont propices à une automatisation massive simplifiée ? Quels sont les langages adaptés pour programmer les équipements réseau ? Quels outils de configuration et de vérification peuvent être utilisés ?

Cet article vise à donner les éléments clés permettant à chaque administrateur réseau de sauter enfin le pas et d'avancer progressivement mais sûrement vers l'automatisation de son réseau.

2 Avant de se lancer dans l'aventure...

Si les statistiques sont si basses concernant l'automatisation des réseaux, il y a des raisons ! Le voyage vers l'automatisation peut être long et semé d'embûches. C'est un changement qui doit s'étudier et se planifier. Mais si le jeu en vaut la chandelle, il est rare de voir les organisations se donner vraiment les moyens d'y arriver.

La première démarche est de choisir entre l'utilisation de solutions d'automatisation « clés en main » ou de construire soi-même sa propre architecture à base d'outils existants et/ou développés en interne.

Dans le premier cas, il faudra principalement évaluer les bénéfices d'un point de vue « métier », et accepter un certain lâcher-prise technologique. A défaut de fonctionner exactement comme vous le souhaiteriez, est-ce que la solution est en adéquation avec les objectifs recherchés ? Est-ce que la solution est adaptée à la taille du réseau ? Quelle flexibilité apporte cette solution d'un point de vue métier ? A quelle vitesse les changements désirés peuvent-ils être implémentés ? Toutes ces questions doivent être traitées en conservant l'objectif final en tête, sans préjuger des lignes de CLI qui semblent a priori nécessaires pour y parvenir. Les solutions clé-en-main suivent généralement un modèle « prescriptif » : elles imposent des technologies et procédures pour retirer au final de la complexité. Cela s'accompagne de-facto de contraintes (renouvellement de matériel, choix d'un constructeur...) dont le coût pourra être facilement évalué et mis en face des bénéfices immédiats.

Dans le cas des approches « développées maison », l'architecte réseau devra faire le travail de transcription des objectifs métier et les lignes de CLI associées. L'avantage certain est que tout est possible à partir du moment où les équipements réseau supportent les fonctions nécessaires. Il n'est pas question ici de subir des contraintes d'un produit avec ses éventuels manques. Par contre, toutes les règles d'automatisation doivent être conçues, développées et testées en long et en large avant la mise en production. La mise en place du projet d'automatisation est donc généralement longue. Les premières étapes simplistes consitent le plus souvent à pousser des canevas (ou « templates ») des configurations les plus fréquemment utilisées.

Figure 1 - Plusieurs approches pour automatiser son réseau

Faire le bon choix nécessite dans les deux cas de se recentrer sur ses réels objectifs, en mettant de côté ses instincts d'expert. Les erreurs typiques sont :

- adopter une solution « clé-en-main » et la personnaliser à tous les niveaux pour la rendre fonctionnelle dans l'organisation. Il est parfois plus efficace de réévaluer la pertinence de certains besoins techniques que de chercher à y répondre directement. Un changement d'approche peut se révéler plus bénéfique que de contourner un modèle prescriptif. Si ni l'un ni l'autre ne sont possibles alors il convient de réfléchir à la mise en place d'une solution « maison ».
- se lancer dans le développement d'une solution « maison » en oubliant les problématiques métiers de l'organisation. Est-ce que créer rapidement des nouveaux VLANs ou déployer quelques blocs de configuration sur les équipements répond bien au problème réel ? Ne faut-il pas travailler sur des problématiques plus complexes/globales ? En laissant de côté les besoins les plus complexes, on conserve la nécessité d'interventions manuelles qui peuvent troubler les configurations poussées automatiquement.

D'une manière générale, il faut absolument prendre le temps d'évaluer les besoins réels, et les pour et les contre de chaque solution, sans aucun préjugé. Dans cet exercice, les experts techniques peuvent facilement se faire piéger en tentant d'automatiser l'application de configurations habituelles bien connues, personnalisées, parfois complexes, sans remettre en perspective leur intérêt pour le métier de l'organisation. L'avènement actuel du SD-WAN illustre parfaitement tout cela. L'intérêt des organisations ne porte pas juste sur l'automatisation du WAN existant, mais sur de nouvelles solutions « clé-en-main » qui répondent réellement aux problématiques concrètes rencontrées. Automatiser l'existant ne suffit pas toujours.

La citation suivante d'Henri Ford résume parfaitement ce paradigme : « If I had asked people what they wanted, they would have said faster horses ». Ce n'est pas parce qu'on opère un réseau au quotidien qu'on est forcément le mieux placé pour savoir ce qui est le plus pertinent. Il ne faut préjuger de rien quand on se lance dans l'aventure de l'automatisation et revenir sur ses besoins primaires !

3 Architectures pour l'automatisation

L'automatisation réseau ne prend tout son sens que si l'on sait homogénéiser un minimum les configurations du réseau. Si tous les équipements ont des configurations différentes (VLANs, ports utilisés, configurations sur les ports d'accès, routage...) le gain de la mise en place d'un outil d'automatisation du réseau sera minime car toute cette diversité nécessitera autant de workflows différents. La complexité ne sera que déplacée au moment de la mise en place d'un outil d'automatisation.

Deux éléments sont indispensables pour réussir son projet d'automatisation :

- la mise en place d'une architecture homogène simple à répliquer,
- et sur les réseaux d'accès, centraliser le contrôle d'accès au réseau en déportant la configuration de la politique d'accès vers un serveur d'authentification.

3.1 Le contrôle d'accès au réseau, levier de l'automatisation

Sur les réseaux d'accès filaires (réseaux de campus ou de sites distants...), les écarts principaux de configuration entre les équipements réseaux résident dans les caractéristiques des ports d'accès. Chaque interface peut connecter des éléments différents avec des configurations différenciées (VLANs distincts, VLAN voix, ACL, QoS...)

S'il est possible de définir des templates pour les types de connexion (Équipe informatique, projet X, étudiant, invité, point d'accès, téléphone...), il reste à la charge de l'administrateur de définir l'association entre ces templates et chaque interface. Ce travail reste laborieux et fait peser un risque sur la sécurité de l'infrastructure réseau car il subsiste un risque d'erreur important dans lequel un utilisateur a soudainement accès à des ressources qui devraient lui être refusées.

C'est tant pour pallier ces problèmes de sécurité que pour éviter ces configurations statiques que l'on assiste depuis plusieurs années à l'explosion du contrôle d'accès réseau. A l'instar de ce qui se fait sur tous les réseaux Wi-Fi, le port d'accès sera banalisé et un serveur d'identité (par exemple FreeRadius [1])

autorisera les connexions et appliquera les configurations appropriées selon l'authentification de l'utilisateur.

Le protocole standard de communication entre un équipement réseau (Authenticator) et un serveur d'identité est RADIUS (RFC 2865) [2]. De nombreuses extensions standards permettent d'appliquer des configurations (VLAN, ACL...) depuis le serveur d'authentification.

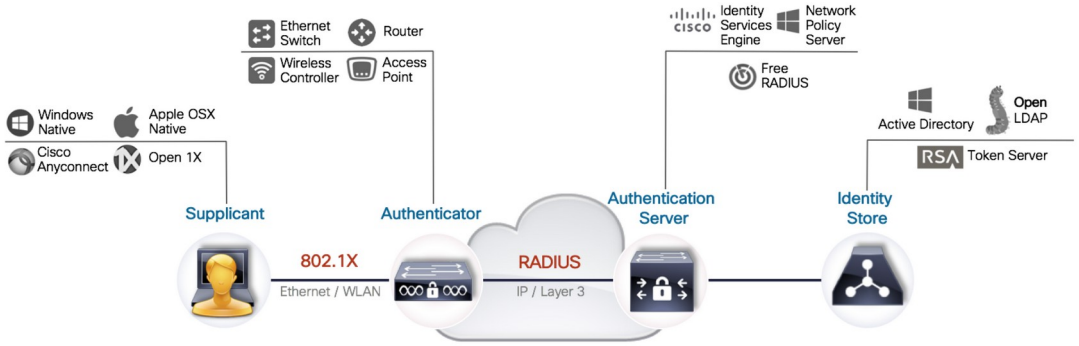

Figure 2 - Composantes du contrôle d'accès

La mise en place du contrôle d'accès réseau permet d'indifférencier les configurations des interfaces d'accès. C'est au niveau du serveur d'authentification que seront appliquées les règles désirées.

Pour réellement pouvoir assurer toutes les configurations depuis le serveur d'authentification, il est nécessaire de pouvoir disposer d'un maximum de souplesse à ce niveau là :

- interfaçage avec un annuaire (type Active Directory ou LDAP),
- interfaçage avec une base d'inventaire regroupant les adresses MAC de certains équipements qui ne peuvent pas s'authentifier. Ici l'adresse MAC sera utilisée pour identifier les matériels (on parle de MAB - MAC Authentication Bypass)

Toujours dans la logique de déporter plus de décisions sur ce serveur d'authentification, on voit sur le marché plus de fonctionnalités permettant de prendre des décisions de manière centralisée :

- Profiling : capacité de détecter le type d'équipement connecté sur analyse DHCP, LLDP, CDP...
- Contrôle de posture : capacité de vérifier si le poste de travail est conforme aux impératifs (ex : vérifier si l'antivirus est actif)
- Interaction avec d'autres logiciels de sécurité pour bannir du réseau des terminaux qui ont été analysés comme étant corrompus.
- ...

D'une manière générale, il convient de privilégier une solution d'authentification ouverte, capable de prendre en charge l'intégralité de la logique de connexion à l'accès, au-delà de la simple gestion d'un 802.1X primaire et d'une authentification sur login/password ou par certificat. Préalablement à la mise en place d'un contrôle d'accès réseau, il convient donc de définir la logique nécessaire permettant de prendre en compte le maximum de terminaux de l'organisation, et de choisir son serveur d'authentification en conséquence.

3.2 Les architectures de fabric

Un élément clé à prendre en compte est l'architecture réseau optimale pour pouvoir automatiser au maximum son infrastructure réseau. Si l'on demande à dix administrateurs réseau de réaliser un schéma

décrivant l'architecture de leur réseau, il est fort probable que l'on obtienne plus de dix résultats différents ! Automatiser est plus complexe dans un monde hétérogène.

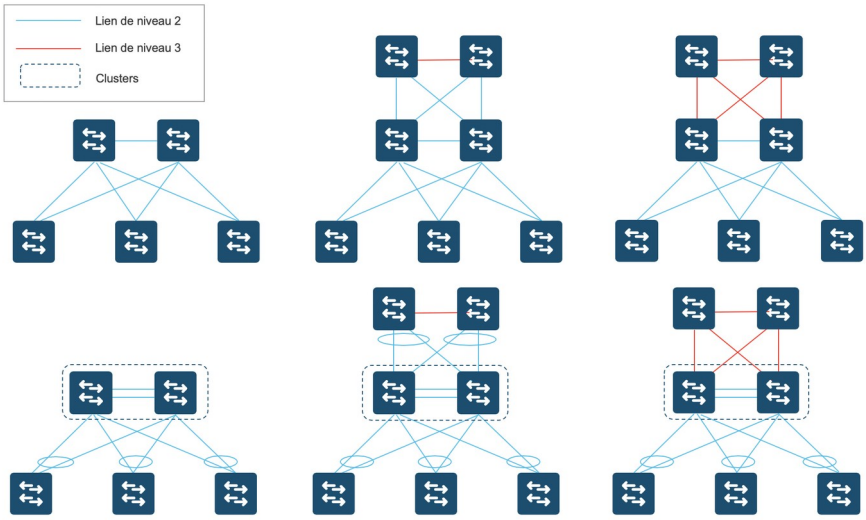

Figure 3 - Exemples de topologies d'accès fréquemment rencontrées

Aussi, dans les réseaux traditionnels, il existe des dépendances fortes entre la topologie logique et la topologie physique. Un sous-réseau IP est associé à un VLAN, configuré sur une partie de l'infrastructure. Le déplacement d'un terminal entraîne généralement la nécessité de changer son adresse IP. Ces dépendances sont à l'encontre de la nécessité d'avoir une configuration homogène sur l'ensemble des équipements du réseau pour mieux pouvoir automatiser leur configuration. En effet, cette dépendance vis-à-vis de la topologie nécessitent de programmer des fonctions parfois complexes avec des actions personnalisées selon l'équipement à configurer.

Pour y remédier, on a pu assister parfois à une extension des VLANs dans certaines organisations. Ces derniers ont été parfois prolongés jusqu'au cœur du réseau, mettant à mal les principes d'architectures à trois niveaux déployés jusque-là. Les niveaux 2 étendus sont souvent à l'origine de boucles, de problématiques spanning-tree et de tempêtes de broadcast.

Depuis quelques années, les technologies de « fabric » ont largement démontré leur intérêt à ce niveau-là. Dans une topologie de fabric, on va décorréler un underlay dont la fonction est de transporter des paquets efficacement entre équipements réseau, et un overlay, identique sur tous les équipements réseau, dont le rôle est d'offrir les services de connectivité. L'underlay et l'overlay étant tous les deux homogènes, à savoir identiques sur les divers équipements réseau, leur automatisation est grandement simplifiée. Le protocole qui tend à s'imposer pour le plan de données est VXLAN [3], permettant un transport de niveau 2 sur IP. L'avantage est que l'underlay est intégralement IP, et repose sur des protocoles de routage fiables, mettant aux oubliettes toutes les problématiques de boucle qui ont mis à mal tant de réseaux Ethernet.

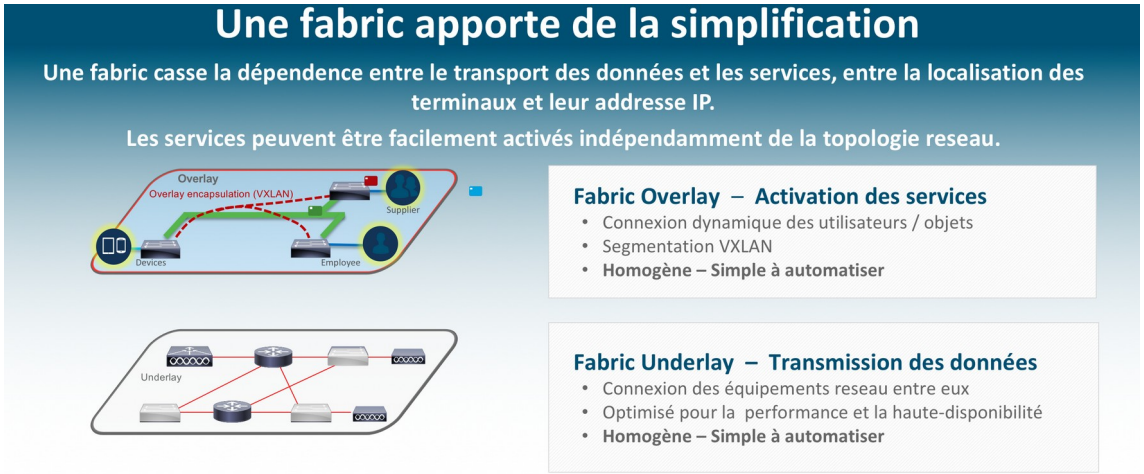

Figure 4 - Décorrélation des plans de transport et de services

En décidant de reposer sur des infrastructures de type fabric, il devient simple d'automatiser la mise en place des services puisqu'il n'y a aucune dépendance avec l'architecture réseau. Tous les équipements deviennent égaux d'un point de vue services et ont de fait des configurations très similaires.

Une fabric n'est pas une fin, mais un moyen !

4 Quels protocoles ?

Les équipements réseau ont initialement été conçus pour être administrés par des humains, experts dans ce domaine. C'est la raison pour laquelle les interfaces CLI (Command Line Interface) ont été développées et sont si puissantes. Elles permettent à un humain de facilement configurer un équipement, via une syntaxe compréhensible, tout en offrant un minimum d'automatisation. En effet, rien de plus simple que de copier des blocs de CLI et de les coller dans plusieurs équipements. Les amateurs des belles GUI (Graphical User Interface) en sont souvent pour leur reste : ils ne peuvent pas aussi facilement répliquer la même opération sur plusieurs équipements sans risquer d'avoir des crampes à force de cliquer sur leur souris !

Si le CLI est optimisé pour l'humain, il convient de s'intéresser à son intérêt dans un cadre d'automatisation réseau dans lequel c'est une machine qui va configurer le réseau :

- Absence d'un réel modèle de données standard (un humain doit lire les guides de configuration)
- Pas de réels codes d'erreur. Il est compliqué de détecter quand une configuration est acceptée ou non et surtout la raison d'un problème
- Les lignes de CLI sont exécutées souvent séquentiellement, et un échec sur une ligne de CLI n'interrompt pas toujours l'action globale de configuration, engendrant potentiellement des incohérences finales
- Risque d'avoir plusieurs configurations simultanées antagonistes sur le même équipement

L'utilisation du CLI par un automate peut tout à fait engendrer le type de situation suivante, cauchemardesque pour l'administrateur qui devra ensuite faire le nécessaire pour rétablir une configuration appropriée sur le réseau.

```
Félicitations !
Votre configuration a été correctement déployée
sur 217 / 276 équipements
```

Toutes ces difficultés et bien d'autres ont été actées dans le RFC 3535 « Overview of the 2002 IAB Network Management Workshop ». C'est le point de départ de la définition d'une interface programmatique pour configurer les équipements réseau : le protocole NETCONF [4].

4.1 NETCONF

NETCONF est un protocole standardisé par l'IETF (RFC 6241) pour le management des réseaux. Il offre de nombreux atouts dont les principaux suivants :

- Capacité de configurer des équipements mais aussi des services
- Mode transactionnel : application de la configuration dans sa globalité et en une fois (annulation de l'ensemble en cas de la moindre erreur)
- Présence de messages d'erreur clairs
- Capacité de configurer les équipements mais aussi de récupérer des variables d'état (par exemple charge des interfaces...)
- Capacité de configurer plusieurs « datastores » (ou configurations distinctes) si les équipements les supportent. Par exemple : running, startup, candidate...
- Possibilité de copier une configuration
- Possibilité de verrouiller une configuration pour éviter que plusieurs configurations antagonistes ne soient faites au même moment.
- Possibilité de « commiter » une configuration candidate pour qu'elle devienne effective sur l'équipement.

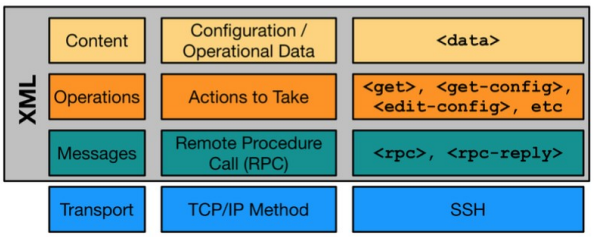

Figure 5 - Structure du Protocole NETCONF

NETCONF permet ainsi de se prémunir de catastrophes illustrées par le message factice montré plus haut. Pour éviter qu'un automate pousse aveuglément des configurations sur des équipements avec des erreurs aléatoires, on pourra faire les opérations suivantes :

- Verrouiller toutes les configurations des équipements
- Modifier toutes les configurations dans les « candidate datastores ». Si une erreur est détectée, annuler la configuration (effacer la candidate configuration)
- Si aucune erreur n'est remontée à ce niveau, cela signifie que les modifications de configuration proposées sont valides sémantiquement. L'automate peut donc « commiter » toutes les modifications qui deviennent immédiatement effectives.

- A ce moment-là chaque équipement (ou serveur dans la terminologie NETCONF) va attendre de recevoir un « commit confirm ». S'il ne le reçoit pas, c'est probablement parce que la configuration poussée est erronée d'un point de vue logique et a cassé la connexion entre l'équipement réseau (le serveur) et l'automate (le client). Si l'équipement ne reçoit pas une confirmation avant un certain temps, alors il va revenir dans la configuration valide précédente.
- L'automate peut en dernier lieu copier le « running datastore » en « startup » afin que les configurations soient persistantes après un redémarrage.

Pouvoir configurer tout à la fois et en mode tout ou rien (c'est-à-dire que le moindre échec annule l'ordre complet de configuration pour éviter les états incertains) est la base de ce que l'on appelle une configuration « transactionnelle ».

Un autre élément très important de NETCONF est la possibilité de connaître les capacités de l'équipement que l'on souhaite configurer. Au moment de l'établissement de la connexion NETCONF, l'équipement réseau indique toutes ses capacités, ou « capabilities » dans la terminologie du protocole. On va distinguer ici les capacités en termes d'opérations (l'équipement va décrire toutes les opérations supportées parmi les exemples donnés plus haut) mais également en termes de modèles de données supportés (c'est-à-dire les configurations qu'il est possible d'effectuer sur l'équipement). Ces modèles sont décrits en utilisant le langage YANG [5].

4.2 YANG

YANG est défini par le RFC IETF 6020. C'est un langage qui permet de définir un modèle de données pour les services réseaux (mais rien n'empêche de l'utiliser pour autre chose). Pour comprendre la notion de modèle de données, on peut par exemple se pencher sur la problématique de description d'un individu. Certains vont peut-être décrire une personne selon l'âge et le sexe, quand d'autres risquent d'utiliser la taille, le poids, la couleur des yeux et des cheveux. Si l'on veut avoir une description cohérente entre plusieurs individus, il faut se mettre d'accord sur les critères que l'on veut renseigner. Il faut se mettre d'accord sur une fiche que l'on va tous remplir selon les mêmes règles. C'est là, l'objet d'un modèle de données, il va tout simplement décrire ce qu'est cette fiche et comment la remplir. YANG est simplement le langage pour réaliser cette description.

Il existe de très nombreux modèles de données qui vont pouvoir décrire plus ou moins tout ce qu'il est possible de configurer sur le réseau : interfaces, BGP, OSPF... Il existe également parfois différents modèles pour décrire la même chose. L'IETF a standardisé de nombreux modèles pour tout ce qui est de son ressort (IP). D'autres modèles sont définis par d'autres organismes comme l'ITU. Un autre groupe intéressant est OpenConfig, qui est un rassemblement d'entreprises ou d'organisations très importants qui ont souhaité imposer leurs modèles selon des critères très opérationnels, sans avoir à subir les processus parfois pénibles des précédents organismes de standardisation. Et bien sûr, chaque constructeur va proposer des modèles propriétaires pour décrire les fonctionnalités les plus avancées ou des protocoles propriétaires pour lesquels les standards n'existent pas encore. Un même équipement réseau va pouvoir supporter plusieurs modèles différents pour spécifier la même chose. Il suffit juste de spécifier celui que l'on utilise.

```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                                        string
   |     +--rw description?                                string
   |     +--rw type                                        identityref
   |     +--rw enabled?                                    boolean
   |     +--[...]
```

Figure 6 - Exemple d'un modèle YANG
(présenté en format « tree » avec l'outil pyang)

A ce stade, il est important de comprendre qu'un modèle ne contient aucune donnée de configuration ou d'état, il représente juste comment ces dernières sont organisées. Il convient également de préciser que les modèles décrivent à la fois des informations de configurations (par exemple l'adresse IP sur une interface) mais également des informations d'état (par exemple : est-ce que l'interface est UP ou DOWN ?).

Le protocole NETCONF décrit précédemment va pouvoir transporter les données en suivant l'organisation décrite dans les différents modèles YANG supportés. Il faut rappeler qu'à l'initialisation d'une session NETCONF, l'équipement réseau (serveur) va indiquer à l'automate (client) tous les modèles YANG supportés. L'automate sait de-facto s'il parle la même langue que l'équipement qu'il doit configurer. Les données dans une connexion NETCONF sont représentées en XML, en suivant exactement un format dérivé directement du modèle YANG. On peut voir ci-dessous une réponse Netconf d'une demande de visibilité de la configuration des interfaces en suivant le modèle ietf-interfaces vu précédemment.

```
<rpc-reply message-id="1"xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
   <interface>
    <name>eth0</name>
    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-
iftype">ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
    [...]
   </interface>
  </interfaces>
 </data>
</rpc-reply>
```

Figure 7 - Exemple d'une réponse NETCONF

Les modèles YANG ne sont pas exclusivement utilisés par NETCONF. D'autres protocoles comme RESTCONF ou gRPC (télémétrie) vont aussi s'appuyer sur des modèles YANG.

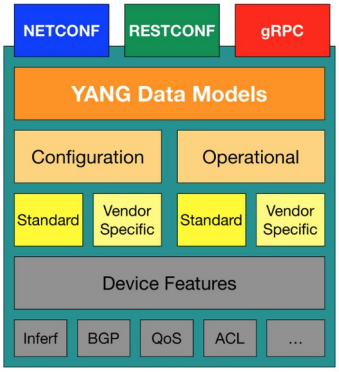

Figure 8 - Structure globale d'un operating system programmable

4.3 RESTCONF

On parle de RESTful API quand les directives fournies sont faites via des directives de type HTTP (GET, POST, DELETE…), La principale caractéristique d'une API RESTful est qu'elle est sans état. En outre, chaque requête correspond à une demande. Il n'y a pas de dialogue possible entre les extrémités de la chaîne via une connexion préalablement établie. On dénote l'API RESTCONF qui permet d'échanger via une API RESTful un modèle YANG. Ces API sont plutôt privilégiées par les communautés de développeurs web et seront plus souvent utilisées dans le contexte d'un datacenter puisque cela correspond au langage naturel de programmation des équipes en charge des applications. Cependant, il convient de noter que RESTCONF n'offre pas toutes les capacités de Netconf, conçu spécifiquement pour l'administration réseau (pas de « transactionnalité », pas d'échange préalable des « capabilities »...)

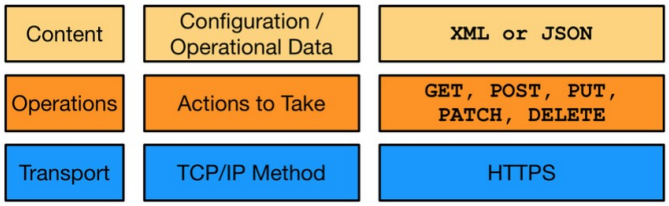

Figure 9 - Structure du Protocole Restconf

Pour résumer, là ou un humain pourra tâtonner pour comprendre comment configurer un équipement à partir de l'aide contextuelle et des guides de configuration, un automate doit s'appuyer sur des interfaces programmatiques fiables et des modèles de données préétablies que l'on va respecter à la lettre pour garantir la fiabilité des opérations.

5 L'automate - Ansible

Il existe de nombreux outils d'automatisation. Des outils propriétaires aux scripts python maison, il serait vain d'essayer de tous les lister. Puppet et Chef ont eu leur heure de gloire, mais ont l'inconvénient de nécessiter des agents sur les équipements managés (faute d'utiliser un protocole de configuration comme NETCONF).

Depuis quelques années, Ansible tend à s'imposer comme outil Open Source d'automatisation. Sa capacité à pouvoir tourner sans nécessité d'agent, sa simplicité, sa richesse en « network modules » supportés, et sa très large communauté d'utilisateurs font d'Ansible un outil particulièrement prisé. S'il est possible de pousser des configurations en CLI, l'outil propose des modules pour configurer certains éléments en

suivant des modèles. Ci-dessous une « TASK » qui utilise le module ios_l2_interface pour configurer une interface layer 2 d'un équipement Cisco IOS.

```
- name: Ensure GigabitEthernet0/5 is a trunk port and ensure 2-50 are being tagged
(doesn't mean others aren't also being tagged)
  ios_l2_interface:
    name: GigabitEthernet0/5
    mode: trunk
    native_vlan: 10
    trunk_vlans: 2-50
```

Figure 10 - Exemple d'utilisation d'un network module Ansible

L'architecte réseau doit définir ce qu'il attend d'Ansible comme actions de configuration. Cela se fait à partir de la création de « playbooks », en utilisant un langage texte simple et lisible par un humain : YAML. L'objectif est que l'humain définisse simplement ce qu'il souhaite faire et laisse les actions complexes à l'automate. Un playbook va enchaîner une suite de « tasks » avec éventuellement des conditions, variables etc. L'administrateur va pouvoir créer des règles d'automatisation de la plus simple (pousser quelques lignes de CLI sur tous les équipements) à la plus compliquée (orchestrer un réseau multi-constructeurs en enchaînant des configurations différentes selon des conditions rencontrées)

Ansible ne s'arrête pas non plus au réseau. De nombreux modules permettent d'échanger avec des systèmes d'exploitation, des systèmes présentant des API REST... La seule limite est l'imagination de l'architecte, le temps dont il dispose et sa capacité à valider chaque playbook.

Ansible peut donc tout à fait pousser des lignes de CLI et configurer des équipements anciens ne fournissant pas d'interfaces programmatiques. Maintenant, dans un contexte d'automatisation, il peut s'avérer plus judicieux de tirer profit de NETCONF et d'éviter des désastres dans le cas ou des tasks sont erronées.

La task suivante, proposée en exemple dans la documentation d'Ansible, va permettre de configurer un serveur NTP sur un équipement. Un modèle YANG IETF étant utilisé, on comprend que cette task va pouvoir être utilisée avec des équipements de plusieurs constructeurs différents à partir du moment où ils supportent cette capacité.

```
- name: set ntp server in the device
  netconf_config:
    content: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
                <ntp>
                    <enabled>true</enabled>
                    <server>
                        <name>ntp1</name>
                        <udp><address>127.0.0.1</address></udp>
                    </server>
                </ntp>
            </system>
```

```
        </config>
```

Figure 11 - Exemple d'une task Ansible utilisant NETCONF

Outre la robustesse apportée par l'utilisation de NETCONF/YANG, (« transactionnalité », codes d'erreur, modèles des données) cela permet de simplifier grandement les playbooks. Il devient inutile de rajouter des conditions appelant des tasks différentes selon les modèles d'équipements.

6 Network as code

Mettre en place un outil d'automatisation ne suffit pas à résoudre tous les problèmes. Une préoccupation première est la capacité à apporter des changements rapidement sans mettre en péril l'infrastructure réseau. Il faut donc prévoir une méthodologie de changement, offrant la capacité de tester et valider les modifications, ainsi que de les archiver. Le rythme des innovations étant tellement important dans les technologies de l'information, il convient d'adopter des méthodes appropriées adaptées à la fois pour les équipes de développement mais aussi pour les équipes opérationnelles. On parle alors de DevOps, qui reflète avant tout d'un changement de mentalité au sein des équipes pour appliquer des procédures qui accélèrent le changement sans mettre en péril le réseau.

Depuis quelques années, le CI/CD (Continuous Integration / Continuous Development) s'impose comme méthode redoutable pour le déploiement logiciel et peut tout à fait s'adapter au réseau. L'intégration et le développement continus permettent d'assurer à la fois le bonheur des architectes qui pourront rapidement apporter des modifications et innovations souhaitées, mais aussi des équipes exploitantes qui seront certaines que chaque modification aura suivi un ensemble de processus et de validations collégialement définies. On parle d'un pipeline CI/CD.

6.1 La gestion des versions - Git

Si l'on revient plus précisément sur l'automatisation permise par Ansible, on peut se préoccuper de la manière dont les nombreux playbooks définis en YAML sont modifiés, potentiellement par plusieurs personnes, et comment ces derniers sont archivés. Les prérequis à ce niveau ne sont absolument pas différents de la gestion d'un code informatique, et des solutions existent depuis longtemps.

Le logiciel de gestion de versions le plus populaire aujourd'hui est Git [6], qui a largement remplacé ses ancêtres CVS et SVN (Subversion). Il est notamment utilisé pour la gestion du code Linux et s'est globalement imposé dans la plupart des développements. Git va permettre de gérer des évolutions de code, la création de versions, de branches logicielles, et permettra à plusieurs architectes de faire des modifications ensemble. Les architectes pourront « commiter » leurs changements et les pousser (« push ») dans le dépôt commun (« repository »). Notons à ce niveau que Github [7] est un dépôt accessible dans le cloud as-a-service. SourceSup [8] est un service de dépôt Git pour la communauté RENATER et sera certainement plus adapté. Notons cependant que l'automatisation réseau est sensible et avec des contraintes opérationnelles telles que la question d'une externalisation doit être mûrement méditée. Il ne s'agirait notamment pas que les login/password de tous les équipements réseau puissent être consultés sur Internet à cause d'un simple raté sur la gestion des droits. Rappelons à ce niveau-là que des automates scannent en continu les principaux dépôts de code pour y découvrir ce type d'informations.

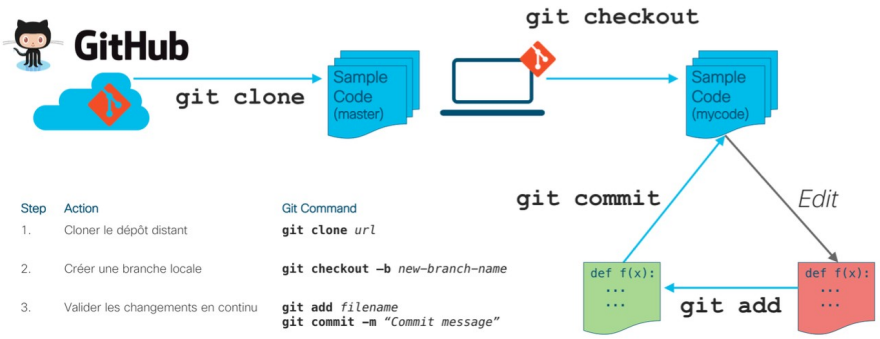

Figure 12 - Opérations classiques de Git (avec GitHub)

Git va donc se montrer infaillible pour permettre de gérer les évolutions de vos règles d'automatisation, pour permettre le travail à plusieurs en conservant une cohérence de l'ensemble, et pourra autoriser des retours arrière quand cela sera nécessaire. En revanche Git ne fait aucun contrôle de la justesse des modifications apportées et ne va pas vérifier leur impact sur le réseau. En d'autres termes, si vous intégrez des changements erronés dans votre version principale, votre réseau risque d'en pâtir. Il faut donc ajouter dans votre pipeline CI/CD un élément qui va tester vos modifications.

6.2 Le test continu des changements - Jenkins

Jenkins [9] est un autre outil Open Source très largement adopté dans un contexte CI/CD. Jenkins va pouvoir s'interfacer directement avec Git et va pouvoir exécuter des tests après chaque modification détectée (grâce à des webhooks depuis le serveur Git).

Tout comme Ansible, Jenkins en soit ne fait rien. C'est juste une boîte à outil dans laquelle vous pourrez définir vos propres méthodes de test, ou « steps » intégrés dans un « Pipeline Jenkins ». L'avantage de Jenkins est la présence d'une interface web conviviale vous permettant d'organiser vos tests, d'en voir les résultats et de définir simplement les actions qui en découlent.

Dans un environnement de développement logiciel, la compilation du code sera une des premières étapes évidentes du pipeline Jenkins. Dans le cadre de l'automatisation réseau avec Ansible, on devra tester les modifications de playbooks et leur conséquence sur le réseau. La mise en place d'un petit réseau de test, physique ou virtuel, représentatif de l'architecture de production est pour cela indispensable. Un environnement de test est une composante essentielle du pipeline CI/CD et il faut la prendre en compte dès le début de la mise en place du projet d'automatisation.

Le pipeline Jenkins intégrera la mise en œuvre des nouveaux playbooks pour ce réseau de test, puis d'autres « steps » pourront être de vérifier que les différents équipements restent joignables. Jenkins affichera les résultats de tous les tests. Le pipeline Jenkins pourra automatiquement intégrer les nouveaux playbooks dans l'infrastructure de production en cas de succès. A vous de définir ce qui est adapté pour votre infrastructure.

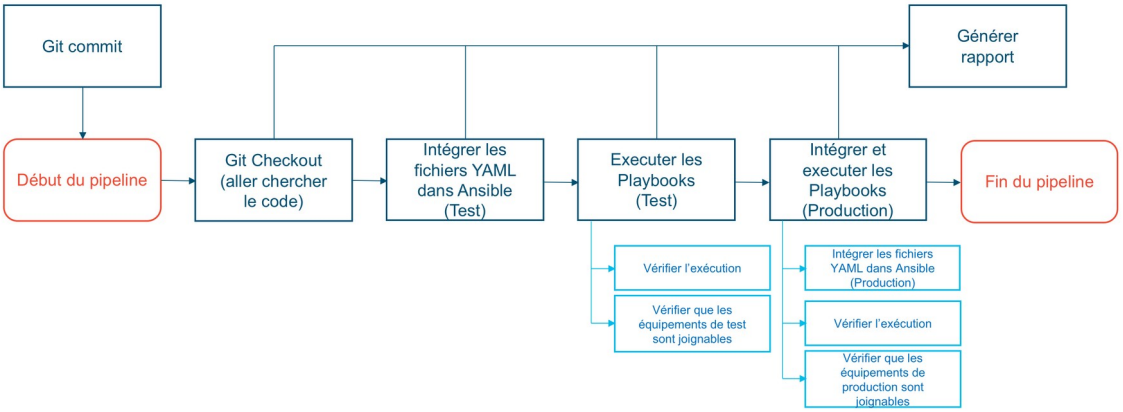

Figure 13 - Exemple simpliste d'un pipeline Jenkins

Il va de soi que le maintien des pipelines Jenkins doit être aussi suivi que celui de la modification des playbooks Ansible. Le processus de validation doit être conservé à niveau pour maintenir la confiance des équipes opérationnelles et conserver la capacité de changement continue.

7 Conclusion

L'automatisation des réseaux est un sujet qui est toujours plus au cœur des préoccupations des organisations. La situation actuelle où la configuration est faite manuellement, équipement par équipement, n'est plus tenable car le nombre d'équipements réseau et leur criticité ont augmenté beaucoup plus vite que les personnels en charge de leur management. Des solutions diverses existent pour répondre aux enjeux de l'automatisation réseau, mais leur mise en place doit être soigneusement réfléchie. Au delà d'outils, c'est toute l'architecture qui est concernée. Un outil qui devra configurer des configurations spécifiques, équipement par équipement, ne sera pas efficace, alors qu'une refonte de l'architecture avec du contrôle d'accès et une architecture de fabric permettra peut-être rapidement d'automatiser et de sécuriser tous les ports d'accès au réseau. De la même manière peut-être découvrirez-vous après deux ans de labeur et de customisation de playbooks Ansible, que vous aurez finalement la même chose qu'une solution fournie clé en main par un éditeur logiciel, témoignant d'un manque d'étude en amont. Enfin, la mise en place d'une automatisation réseau ne peut pas se faire sans un minimum d'outillage permettant de tester les changements afin de permettre de répondre à la première demande : avoir un réseau fonctionnel ! Bien évidemment, il est impossible de tout faire en un jour et le voyage vers l'automatisation doit être décomposé en de multiples étapes avec à chaque fois un gain opérationnel clair, mais il convient de ne pas perdre de vue une cible globale.

Il faut également rappeler que l'automatisation n'est qu'une des composantes de l'administration des réseaux informatiques. Les préoccupations des équipes opérationnelles sont tout autant sur la sécurité des infrastructures et la capacité à mieux détecter les problèmes et à y remédier. Automatiser un réseau n'a aucun sens si l'on ne travaille pas en parallèle sur ces deux axes tout autant critiques.

Bibliographie

1: Freeradius web page,https://freeradius.org/

2: C. Rigney, S. Willens, A. Rubens, W. Simpson, RFC 2865 - Remote Authentication Dial In User Service (RADIUS), 2000

3: M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright, RFC 7348 - Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, 2014

4: R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, RFC 6241 - Network Configuration Protocol (NETCONF), 2011

5: M. Bjorklund, RFC 6020 - YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), 2010

6: GIT web page,https://www.git-scm.com/

7: Github web page,https://www.github.com

8: Sourcesup web page,https://services.renater.fr/sourcesup/index

9: Jenkins web page,https://jenkins.io