Équipe d'ingénierie de l'Internet (IETF)
Request for Comments : 8342
RFC mise à jour : 7950
Catégorie : Sur la voie de la normalisation
ISSN: 2070-1721
Traduction Claude Brière de L'Isle

M. Bjorklund, Tail-f Systems
J. Schoenwaelder, Jacobs University
P. Shafer & K. Watsen, Juniper Networks
R. Wilton, Cisco Systems
mars 2018

Architecture de magasin de données de gestion de réseau (NMDA)

Résumé
Les magasins de données sont un concept fondamental qui lie les modèles de données écrits dans le langage de modélisation de données YANG pour les protocoles de gestion de réseau tels que le protocole de configuration de réseau (NETCONF, *Network Configuration Protocol*) et RESTCONF. Le présent document définit un cadre architectural pour les magasins de données fondé sur l'expérience qu'on en a depuis le modèle initial plus simple, en répondant aux exigences qui n'étaient pas très bien prises en charge dans le modèle initial. Le présent document met à jour la RFC 7950.

Statut de ce mémoire
Ceci est un document de l'Internet sur la voie de la normalisation.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Tous les documents approuvés par l'IESG ne sont pas candidats à devenir une norme de l'Internet ; voir la Section 2 de la RFC7841.

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc8342.

Table des Matières

1. Introduction

Le présent document fournit un cadre architectural pour les magasins de données tels qu'ils sont utilisés par les protocoles de gestion de réseau comme le protocole de configuration de réseau (NETCONF, *Network Configuration Protocol*) [RFC6241], RESTCONF [RFC8040], et le langage de modélisation de données YANG [RFC7950]. Les magasins de données sont un concept fondamental qui lie les modèles de données de gestion de réseau aux protocoles de gestion de réseau. L'accord sur un modèle architectural commun de magasins de données assure que les modèles de données peuvent être écrits d'une façon neutre à l'égard du protocole de gestion de réseau. Ce cadre architectural identifie un ensemble de magasins de données conceptuels, mais ne rend pas obligatoire que tous les protocoles de gestion de réseau exposent tous ces magasins de données conceptuels. Cette architecture est neutre à l'égard du codage utilisé par les protocoles de gestion de réseau.

Le présent document met à jour la RFC 7950 en précisant la définition de l'arborescence accessible pour certains contextes du langage de chemin XML (XPath) (voir au paragraphe 6.1) le contexte d'invocation des opérations (voir au paragraphe 6.2).

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119], [RFC8174] quand, et seulement quand ils apparaissent tout en majuscules, comme montré ci-dessus.

2. Objectifs

Les objets de données de gestion de réseau peuvent souvent prendre deux valeurs différentes : la valeur configurée par l'utilisateur ou une application (configuration) et la valeur que l'appareil utilise réellement (état de fonctionnement). Ces deux valeurs peuvent être différentes pour un certain nombre de raisons, par exemple, des interactions internes au système avec le matériel, des interactions avec les protocoles ou avec d'autres appareils, ou simplement le temps que cela prend de propager un changement de configuration aux composants de logiciel et de matériel d'un système. De plus, les objets de données de configuration et d'état de fonctionnement peuvent avoir des durées de vie différentes.

Le modèle originel de magasins de données exigeait que ces objets de données soient modélisés deux fois dans le schéma YANG – comme objets "config true" et comme objets "config false". La convention adoptée par le modèle de données d'interfaces [RFC8343] et le modèle de données IP [RFC8344] était d'utiliser deux branches séparées prenant leur source à la racine de l'arborescence des données : une branche pour les objets de données de configuration et une branche pour les objets de données d'état de fonctionnement.

La duplication des définitions et la séparation ad hoc des données d'état de fonctionnement des données de configuration pose un certain nombre de problèmes. Avoir les données de configuration et d'état de fonctionnement dans des branches séparées du modèle de données complique le fonctionnement et impacte la lisibilité des définitions de modules. De plus, la relation entre les branches n'est pas lisible par la machine, et les expressions de filtres fonctionnant sur la configuration et l'état de fonctionnement qui s'y rapporte sont différentes.

Avec le modèle architectural révisé des magasins de données défini dans le présent document, les objets de données ne sont définis qu'une seule fois dans le schéma YANG mais des instanciations indépendantes peuvent apparaître dans des magasins de données différents, par exemple, une pour une valeur configurée et une autre pour une valeur utilisée dans le fonctionnement. Cela donne une solution plus élégante et simple au problème.

Le modèle architectural révisé de magasins de données prend en charge des magasins de données supplémentaires pour les systèmes qui prennent en charge des chaînes de traitement plus avancées qui convertissent la configuration en état de

fonctionnement. Par exemple, certains systèmes prennent en charge une configuration qui n'est pas actuellement utilisée (appelée une "configuration inactive") ou bien prennent en charge des gabarits de configuration qui sont utilisés pour étendre des données de configuration via un gabarit commun.

3. Terminologie

Le présent document définit la terminologie suivante. Certains des termes sont des définitions révisées de termes définis à l'origine dans la [RFC6241] et la [RFC7950] (voir aussi la Section 4). Les définitions révisées sont sémantiquement équivalentes aux définitions qu'on trouve dans la [RFC6241] et la [RFC7950]. Il est prévu que les définitions révisées fournies dans cette section remplacent les définitions de la [RFC6241] et de la [RFC7950] lorsque ces documents seront révisés.

magasin de données : lieu conceptuel pour mémoriser et accéder aux informations. Un magasin de données peut être mis en œuvre, par exemple, en utilisant des fichiers, une base de données, des localisations de mémoire flash, ou des combinaisons de ceux-ci. Un magasin de données se transpose en une instanciation d'arborescence de données YANG.

nœud de schéma : un nœud dans l'arborescence de schéma. La définition formelle est fournie dans la RFC 7950.

schéma de magasin de données : ensemble combiné de nœuds de schéma pour tous les modules pris en charge par un magasin de données particulier, en prenant en considération toutes les déviations et caractéristiques activées pour ce magasin de données.

configuration : données qui sont requises pour faire passer un appareil de son état initial par défaut à l'état de fonctionnement désiré. Ces données sont modélisées dans YANG en utilisant des nœuds "config true". La configuration peut avoir pour origine des sources différentes.

magasin de données de configuration : magasin de données qui détient la configuration.

magasin de données de configuration en cours : magasin de données de configuration qui contient la configuration actuelle de l'appareil. Cela peut inclure une configuration qui exige des transformations supplémentaires avant qu'elle puisse être appliquée. Ce magasin de données est désigné comme "<running>" *(en cours)*.

magasin de données de configuration candidat : magasin de données de configuration qui peut être manipulé sans impacter le magasin de données de configuration en cours de l'appareil et qui peut être affecté au magasin de données de configuration en cours. Ce magasin de données est désigné comme "<candidate>".

magasin de données de configuration de démarrage : magasin de données de configuration qui détient la configuration chargée par l'appareil dans le magasin de données de configuration en cours lorsque il s'amorce. Ce magasin de données est désigné comme "<startup>" *(démarrage)*.

configuration prévue : configuration qui est destinée à être utilisée par l'appareil. Elle représente la configuration après que toutes les transformations de configuration en <running> ont été effectuées et c'est la configuration que le système tente d'appliquer.

magasin de données de configuration prévu : magasin de données de configuration qui détient la configuration complète prévue de l'appareil. Ce magasin de données est désigné comme "<intended>".

transformation de configuration : ajout, modification, ou suppression de configuration entre les magasins de données <running> et <intended>. Des exemples de transformations de configuration incluent la suppression de configuration inactive et la configuration produite par l'expansion de gabarits.

magasin de données de configuration conventionnel : un des ensembles suivants de magasins de données de configuration : <running>, <startup>, <candidate>, et <intended>. Ces magasins de données partagent un schéma de magasin de données commun, et les opérations de protocole permettent de copier des données entre ces magasins de données. Le terme "conventionnel" est choisi comme terme générique pour ces magasins de données.

configuration conventionnelle : configuration qui est mémorisée dans tout magasin de données de configuration conventionnel.

magasin de données de configuration dynamique : magasin de données de configuration détenant la configuration obtenue de façon dynamique durant le fonctionnement d'un appareil à travers une interaction avec d'autres systèmes, plutôt qu'à

travers un des magasins de données de configuration conventionnels.

configuration dynamique : configuration obtenue via un magasin de données de configuration dynamique.

configuration apprise : configuration qui a été apprise via des interactions de protocole avec d'autres systèmes et qui n'est une configuration ni conventionnelle ni dynamique.

configuration système : configuration qui est fournie par l'appareil lui-même.

configuration par défaut : configuration qui n'est pas fournie explicitement mais pour laquelle est utilisée une valeur définie dans le modèle de données.

configuration appliquée : configuration qui est utilisée de façon active par un appareil. La configuration appliquée a son origine dans une configuration conventionnelle, dynamique, apprise, système, et par défaut.

état du système : données supplémentaires sur un système qui ne sont pas de configuration, comme des informations en lecture seule et des statistiques collectées. L'état du système est transitoire et est modifié par des interactions avec des composants internes et d'autres systèmes. L'état du système est modélisé dans YANG en utilisant les nœuds "config false".

état de fonctionnement : combinaison de la configuration appliquée et de l'état du système.

magasin de données d'état de fonctionnement : magasin de données détenant l'état de fonctionnement complet de l'appareil. Ce magasin de données est désigné comme "<operational>".

origine : annotation de méta données indiquant l'origine d'un élément de données.

configuration rémanente : configuration qui reste partie de la configuration appliquée pour un temps après qu'elle a été supprimée de la configuration prévue ou de la configuration dynamique. La durée peut être minimale ou peut durer jusqu'à ce que toutes les ressources utilisées par la configuration récemment supprimée (par exemple, des connexions réseau, des allocations de mémoire, des brides de fichier) aient été désallouées.

Les termes supplémentaires suivants ne sont pas spécifiques des magasins de données, mais ils sont couramment utilisés et donc définis aussi ici :

client : entité qui peut accéder à des données définies par YANG sur un serveur, sur un protocole de gestion de réseau.

serveur : entité qui fournit à un client l'accès à des données définies par YANG, sur un protocole de gestion de réseau.

notification : message initié par un serveur qui indique qu'un certain événement a été reconnu par le serveur.

appel de procédure distant : opération qui peut être invoquée par un client sur un serveur.

4. Fondements

NETCONF [RFC6241] donne les définitions suivantes :

- magasin de données : lieu conceptuel pour mémoriser et accéder à des informations. Un magasin de données peut être mis en œuvre, par exemple, en utilisant des fichiers, une base de données, des localisations de mémoire flash, ou une de leurs combinaisons.

- magasin de données de configuration : magasin de données qui détient l'ensemble complet de configuration qui est requis pour faire passer un appareil de son état initial par défaut à l'état de fonctionnement désiré.

YANG 1.1 [RFC7950] donne les précisions suivantes lorsque NETCONF est utilisé avec YANG (ce qui est le cas usuel, mais noter que NETCONF a été défini avant l'existence de YANG) :

- magasin de données : lorsque modélisé avec YANG, un magasin de données est réalisé comme une instance d'arborescence de données.

- magasin de données de configuration : lorsque modélisé avec YANG, un magasin de données de configuration est

réalisé comme une instance d'arborescence de données avec configuration.

[RFC6244] définit les données d'état de fonctionnement comme suit :

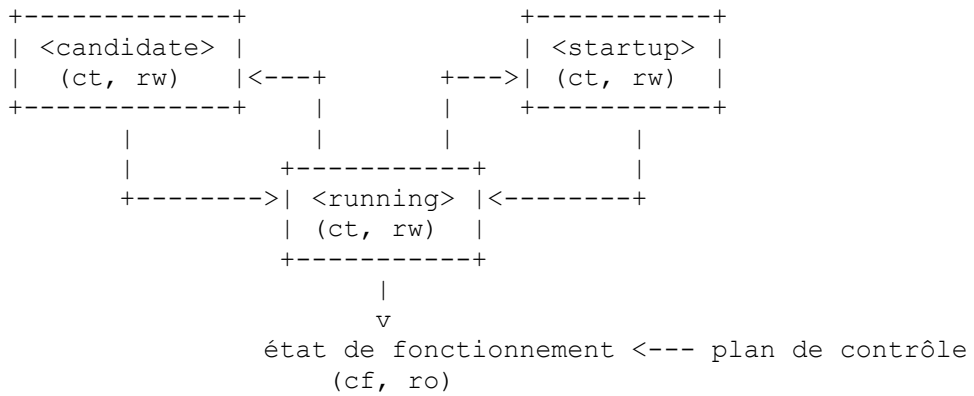
o les données d'état de fonctionnement sont un ensemble de données qui ont été obtenues par le système au démarrage et qui influence le comportement du système de façon similaire aux données de configuration. À la différence des données de configuration, l'état de fonctionnement est transitoire et est modifié par les interactions avec les composants internes ou les autres systèmes via des protocoles spécialisés.

Le paragraphe 4.3.3 de la [RFC6244] discute de l'état de fonctionnement et mentionne, entre autres choses, l'option de considérer l'état de fonctionnement comme étant mémorisé dans un autre magasin de données. Le paragraphe 4.4 de la [RFC6244] conclut alors que, au moment de sa rédaction, l'état de modélisation comme feuilles distinctes et branches distinctes est l'approche recommandée.

L'expérience de mise en œuvre et les demandes des opérateurs [OpState-Reqs] [OpState-Modeling] indiquent que le modèle du magasin de données conçu initialement pour NETCONF et précisé par YANG doit être étendu. En particulier, les notions de configuration prévue et de configuration appliquée ont été développées.

4.1 Modèle originel de magasin de données

Le dessin qui suit montre le modèle original de magasin de données tel qu'il est actuellement utilisé par NETCONF [RFC6241] :



ct = config true ; cf = config false
rw = read-write ; ro = read-only
Les boîtes notent les magasins de données

Figure 1

Noter que ce diagramme simplifie le modèle : "read-only" (ro) et "read-write" (rw) doivent être compris du point de vue du client, à un niveau conceptuel. Dans NETCONF, par exemple, la prise en charge de <candidate> et de <startup> est facultative, et <running> n'a pas à être en écriture. De plus, <startup> peut seulement être modifié en copiant <running> en <startup> dans le modèle d'édition de magasin de données standard NETCONF. Le protocole RESTCONF ne présente pas ces différences et fournit seulement à la place un magasin de données unifié en écriture, qui cache si l'édition est faite par <candidate>, en modifiant directement <running>, ou via un autre mécanisme spécifique de mise en œuvre. RESTCONF cache aussi comment la configuration est rendue persistante. Noter que les mises en œuvre peuvent aussi avoir des magasins de données supplémentaires qui peuvent propager des changements à <running>. NETCONF mentionne explicitement des "magasins de données désignés".

Observations :

o L'état de fonctionnement n'a pas été défini comme un magasin de données, bien qu'il y ait eu des propositions pour introduire un magasin de données d'état de fonctionnement.

o L'opération NETCONF <get> retourne le contenu de <running> avec l'état de fonctionnement. Il est donc nécessaire que les données "config false" soient dans une branche différente des données "config true" si l'état de fonctionnement peut avoir une durée de vie différente de celle de la configuration ou si la configuration n'est pas appliquée immédiatement ou avec succès.

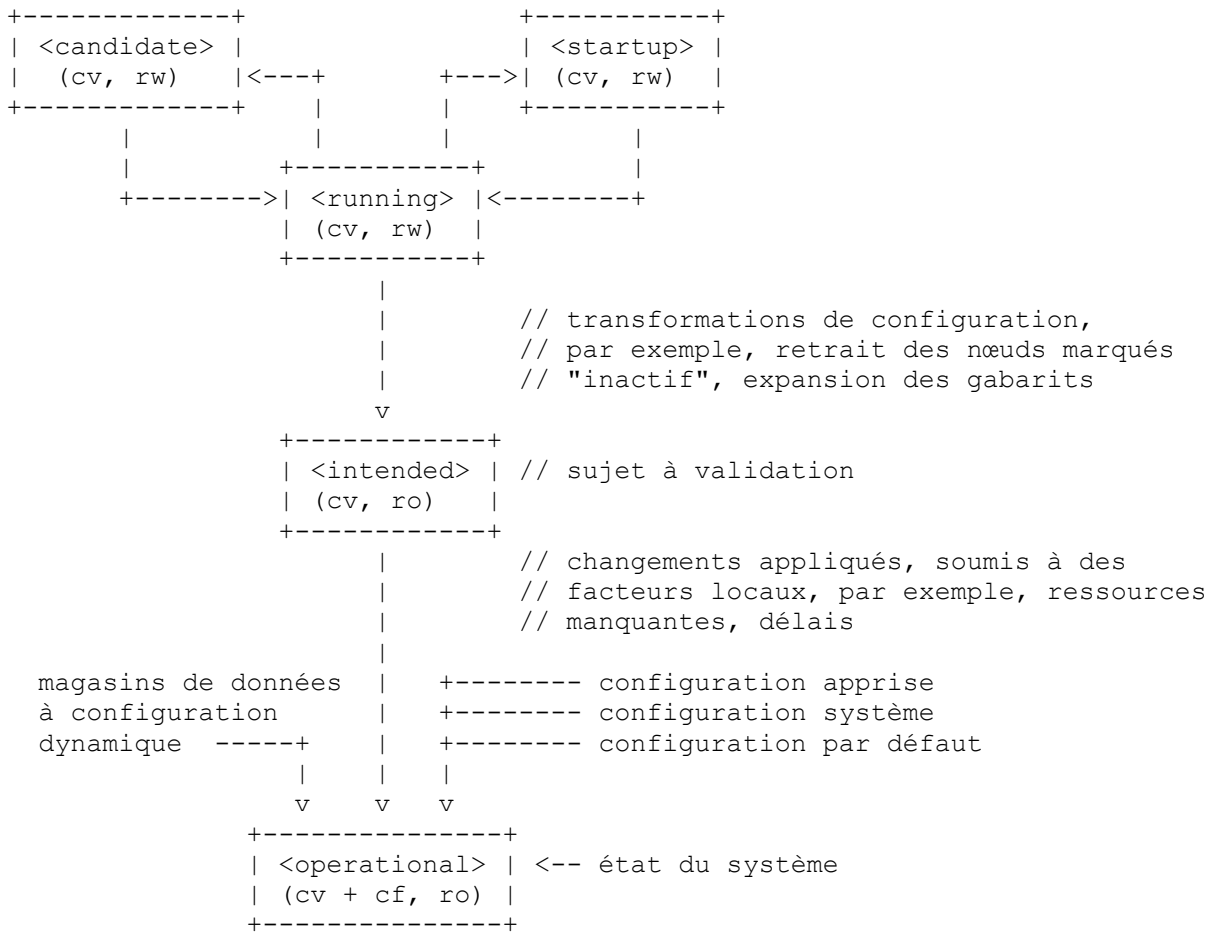
- Plusieurs mises en œuvre ont des mécanismes propriétaires qui permettent aux clients de mémoriser les données inactives dans <running>. Les données inactives sont conceptuellement supprimées avant la validation.

- Certaines mises en œuvre ont des mécanismes propriétaires qui permettent aux clients de définir des gabarits de configuration dans <running>. Ces gabarits sont développés automatiquement par le système, et la configuration résultante est appliquée en interne.

- Certains opérateurs ont rapporté qu'il est essentiel pour eux d'être capables de restituer la configuration qui a en fait été appliquée avec succès, qui peut être un sous ensemble ou un super ensemble de la configuration <running>.

5. Modèle d'architecture de magasin de données

Un nouveau modèle conceptuel de magasins de données est présenté ci-dessous, qui étend le modèle original afin de refléter l'expérience acquise sur le modèle d'origine.



cv = config vraie ; cf = config fausse
rw = lecture-écriture ; ro = lecture-seule
Les boîtes notent les magasins de données désignés

Figure 2

5.1. Magasin de données de configuration conventionnel

Les magasins de données de configuration conventionnels sont un ensemble de magasins de données de configuration qui partagent exactement le même schéma de magasin de données, permettant aux données d'être copiées entre eux. Le terme est vu comme une description parapluie générique de ces magasins de données. Si un module ne contient aucun nœud de données de configuration et s'il n'est pas nécessaire pour satisfaire des importations, il PEUT alors être omis du schéma de magasin de données pour les magasins de données de configuration conventionnels. L'ensemble des magasins de données

inclut :
o <running> *(en cours)*
o <candidate> *(candidat)*
o <startup> *(démarrage)*
o <intended> *(prévu)*

D'autres magasins de données de configuration conventionnels pourront être définis dans de futurs documents.

Les flux de données entre ces magasins de données sont décrits à la Section 5.

Les protocoles spécifiques peuvent définir des opérations explicites pour faire des copies entre ces magasins de données, par exemple, NETCONF définit l'opération <copy-config>.

5.1.1 Magasin de données de configuration de démarrage (<startup>)

Le magasin de données de configuration de démarrage (<startup>) est un magasin de données de configuration qui détient la configuration chargée par l'appareil à l'amorçage. <startup> n'est présent que sur les appareils qui séparent la configuration de démarrage du magasin de données de configuration en cours.

Le magasin de données de configuration de démarrage peut n'être pas pris en charge par tous les protocoles ou toutes les mises en œuvre.

Sur les appareils qui prennent en charge la mémorisation non volatile, le contenu de <startup> va normalement persister à travers les réamorçages via cette mémorisation. Au moment de l'amorçage, l'appareil charge la configuration de démarrage mémorisée dans <running>. Pour sauvegarder une nouvelle configuration de démarrage, les données sont copiées dans <startup> via une opération de protocole implicite ou explicite.

5.1.2 Magasin de données de configuration candidat (<candidate>)

Le magasin de données de configuration candidat (<candidate>) est un magasin de données de configuration qui peut être manipulé sans impacter la configuration courante de l'appareil et qui peut être affecté à <running>.

Le magasin de données de configuration candidat ne peut pas être pris en charge par tous les protocoles ou mises en œuvre.

<candidate> ne persiste normalement pas à travers les réamorçages, même en présence de mémorisation non volatile. Si <candidate> est mémorisé en utilisant une mémorisation non volatile, il est rétabli au moment du réamorçage au contenu de <running>.

5.1.3 Magasin de données de configuration en cours (<running>)

Le magasin de données de configuration en cours (<running>) est un magasin de données de configuration qui détient la configuration en cours de l'appareil. Il PEUT inclure une configuration qui exige d'autres transformations avant qu'elle puisse être appliquée, par exemple, configuration inactive, ou configuration fondée sur un mécanisme de gabarit qui a besoin d'une expansion supplémentaire. Cependant, <running> DOIT toujours être une arborescence de données de configuration valides, comme défini au paragraphe 8.1 de la [RFC7950].

<running> DOIT être pris en charge si l'appareil peut être configuré via des magasins de données de configuration conventionnels.

Si un appareil n'a pas disponibles de mémorisations <startup> et non volatile distinctes, l'appareil va normalement utiliser cette mémorisation non volatile pour permettre que <running> persiste à travers les réamorçages.

5.1.4 Magasin de données de configuration prévu (<intended>)

Le magasin de données de configuration prévu (<intended>) est un magasin de données de configuration en lecture seule. Il représente la configuration après que toutes les transformations de configuration en <running> sont effectuées (par exemple, expansion de gabarit, suppression de configuration inactive) et est la configuration que le système tente d'appliquer.

<intended> est étroitement couplé à <running>. Chaque fois que des données sont écrites dans <running>, le serveur DOIT aussi immédiatement mettre à jour et valider <intended>.

<intended> PEUT aussi être mis à jour indépendamment de <running> si l'effet d'une transformation de configuration change, mais <intended> DOIT toujours être une arborescence de données de configuration valides, comme défini au paragraphe 8.1 de la [RFC7950].

Pour des mises en œuvre simples, <running> et <intended> sont identiques.

Le contenu de <intended> est aussi en relation avec le sous ensemble "config true" de <operational> ; donc, un client peut déterminer dans quelle mesure la configuration prévue est actuellement utilisée en vérifiant si le contenu de <intended> apparaît aussi dans <operational>.

<intended> ne persiste pas à travers les réamorçages ; sa relation avec <running> rend cela inutile.

Actuellement, il n'y a pas de mécanisme standard défini qui affecte <intended> afin qu'il ait un contenu différent de <running>, mais cette architecture permet qu'un tel mécanisme soit défini.

Un exemple d'un tel mécanisme est la prise en charge du marquage de nœuds comme inactifs dans <running>. Les nœuds inactifs ne sont pas copiés dans <intended>. Un second exemple est la prise en charge des gabarits, qui peuvent effectuer des transformations sur la configuration à partir de <running> à la configuration écrite en <intended>.

5.2 Magasin de données de configuration dynamiques

Le modèle reconnaît le besoin de configuration dynamique de magasins de données qui ne font, par définition, pas partie de la configuration persistante d'un appareil. Dans certains contextes, ceux-ci ont été appelés "magasins de données éphémères", car les informations sont éphémères, c'est-à-dire, perdues au réamorçage. Les magasins de données à configuration dynamique interagissent avec le reste du système grâce à <operational>.

Le schéma de magasin de données pour un magasin de données de configuration dynamique PEUT différer du schéma de magasin de données utilisé pour les magasins de données de configuration conventionnels. Si un module ne contient aucun nœud de données de configuration et s'il n'est pas nécessaire pour satisfaire des importations, il PEUT alors être omis du schéma de magasin de données pour un magasin de données de configuration dynamique.

5.3 Magasin de données en état de fonctionnement (<operational>)

Le magasin de données en état de fonctionnement (<operational>) est un magasin de données en lecture seule qui consiste en nœuds tout en "config true" et "config false" définis dans le schéma de magasin de données. Dans le modèle original NETCONF, l'état de fonctionnement a seulement des nœuds "config false". La raison de l'incorporation ici des nœuds "config true" est d'être capable d'exposer tous les réglages opérationnels sans avoir à dupliquer les définitions dans les modèles de données.

<operational> contient l'état du système et toute la configuration actuellement utilisée par le système. Cela inclut toute la configuration appliquée à partir de <intended>, de la configuration apprise, de la configuration fournie par le système, et les valeurs par défaut définies par tout modèle de données pris en charge. De plus, <operational> contient aussi la configuration appliquée à partir des magasins de données à configuration dynamique.

Le schéma de magasin de données pour <operational> DOIT être un super ensemble des schémas combinés de magasin de données utilisés dans tous les magasins de données de configuration, excepté que les nœuds de données de configuration pris en charge dans un magasin de données de configuration PEUVENT être omis dans <operational> si un serveur n'est pas capable d'en faire précisément rapport.

Les demandes de restitution des nœuds de <operational> retournent toujours la valeur utilisée si le nœud existe, sans considération de toute valeur par défaut spécifiée dans le module YANG. Si aucune valeur n'est retournée pour un certain nœud, cela implique alors que le nœud n'est pas utilisé par l'appareil.

L'interprétation de ce qui constitue "être utilisé" par le système dépend de la définition du schéma et de la mise en œuvre de l'appareil. Généralement, les fonctionnalités qui sont activées et opérationnelles sur le système vont être considérées comme "utilisées". À l'inverse, les fonctionnalités qui ne sont ni activées ni opérationnelles sur le système sont considérées comme n'étant pas "utilisées" ; donc, elles DEVRAIENT être omises de <operational>.

<operational> DEVRAIT se conformer à toutes les contraintes spécifiées dans le modèle de données, mais étant donné le but principal de retourner les valeurs "utilisées", il est possible que des contraintes soient violées dans certaines

circonstances (par exemple, une valeur anormale est "utilisée", la structure d'une liste est modifiée, ou une configuration rémanente (voir au paragraphe 5.3.1) existe encore). Noter que des déviations DEVRAIENT être utilisées lorsque on sait à l'avance qu'un appareil ne se conforme pas pleinement au schéma <operational>.

Seules les contraintes sémantiques PEUVENT être violées. Ce sont les déclarations YANG "when", "must", "mandatory", "unique", "min-elements", et "max-elements" ; et l'unicité des valeurs clés.

Les contraintes syntaxiques NE DOIVENT PAS être violées, incluant l'organisation hiérarchique, les identifiants, et les contraintes fondées sur le type. Si un nœud en <operational> ne satisfait pas aux contraintes syntaxiques, il NE DOIT PAS être retourné, et un autre mécanisme devrait être utilisé pour marquer l'erreur.

<operational> ne persiste pas à travers les réamorçages.

5.3.1 Configuration rémanente

Les changements de configuration peuvent prendre du temps pour se diffuser jusqu'à <operational>. Durant cette période, <operational> peut contenir des nœuds pour les deux configurations, précédente et courante, retraçant d'aussi près que possible le fonctionnement courant de l'appareil. Une telle configuration rémanente de la configuration précédente persiste jusqu'à ce que le système ait libéré les ressources utilisées par la configuration qui vient d'être supprimée (par exemple, des connexions réseau, des allocations de mémoire, des rappels de fichiers).

La configuration rémanente est un exemple courant de cas où les contraintes de sémantique définies dans le modèle de données ne peuvent pas être fiables pour <operational>, car le système peut avoir une configuration rémanente dont les contraintes étaient valides avec la configuration précédente et ne sont plus valides avec la configuration courante. Comme les contraintes sur les nœuds "config false" peuvent se référer aux nœuds "config true", une configuration rémanente peut forcer la violation de ces contraintes.

5.3.2 Ressources manquantes

La configuration dans <intended> peut se référer à des ressources qui ne sont pas disponibles ou autrement non présentes physiquement. Dans ces situations, ces parties de <intended> ne sont pas appliquées. Les données apparaissent dans <intended> mais n'apparaissent pas dans <operational>.

Un exemple typique est une configuration d'interface qui se réfère à une interface qui n'est pas actuellement présente. Dans une telle situation, la configuration d'interface reste dans <intended> mais la configuration d'interface ne va pas apparaître dans <operational>.

Noter que la validité de la configuration ne peut pas dépendre de l'état en cours de telles ressources, car cela impliquerait que retirer une ressource pourrait rendre la configuration invalide. Cela est inacceptable, en particulier parce que le réamorçage d'un tel appareil le ferait redémarrer avec une configuration invalide. On permet plutôt que la configuration pour des ressources manquantes existe dans <running> et <intended>, mais qu'elle n'apparaissent pas dans <operational>.

5.3.3 Ressources contrôlées par le système

Parfois, des ressources sont contrôlées par l'appareil et les données correspondantes contrôlées par le système apparaissent dans (et disparaissent de) <operational> de façon dynamique. Si une ressource contrôlée par le système a une configuration correspondante dans <intended> quand elle apparaît, le système va essayer d'appliquer la configuration ; cela cause l'apparition éventuelle de la configuration dans <operational> (si l'application de la configuration est réussie).

5.3.4 Annotation de métadonnées d'origine

Lorsque la configuration passe à <operational>, elle est conceptuellement marquée avec une annotation de métadonnées [RFC7952] qui indique son origine. L'origine s'applique à tous les nœuds de configuration sauf aux conteneurs de non présence. L'annotation de métadonnées "origin" est définie à la Section 7. Les valeurs sont des identités YANG. Les identités suivantes sont définies :

origin : identité de base abstraite d'où les autres identités d'origine sont déduites.

intended : représente la configuration fournie par <intended>.

dynamic : représente la configuration fournie par un magasin de données de configuration dynamique.

system : représente la configuration fournie par le système lui-même. Des exemples de configuration système incluent une configuration appliquée pour une interface de bouclage toujours existante, ou une configuration d'interface qui est auto créée du fait du matériel actuellement présent dans l'appareil.

learned : représente la configuration qui a été apprise via des interactions de protocole avec d'autres systèmes, incluant des protocoles comme les négociations de couche de liaison, des protocoles d'acheminement, et DHCP.
default : représente une configuration qui utilise une valeur par défaut spécifiée dans le modèle de données, utilisant soit des valeurs dans la déclaration "default", soit toute valeur décrite dans la déclaration "description". L'origine par défaut n'est utilisée que lorsque la configuration n'a pas été fournie par une autre source.
unknown : représente une configuration pour laquelle le système ne peut pas identifier l'origine.

Ces identités peuvent être encore précisées, par exemple, il peut y avoir des identités séparées pour des types ou instances particuliers de magasins de données à configuration dynamique dérivés de "dynamic".

Pour tous les nœuds de données de configuration dans <operational>, l'appareil DEVRAIT rapporter l'origine qui reflète le plus précisément la source de la configuration qui est utilisée par le système.

Dans les cas où il pourrait y avoir une ambiguïté sur l'origine qui devrait être utilisée, c'est-à-dire, lorsque la même valeur de nœud de données a été générée de plusieurs sources, la déclaration "description" dans le module YANG DEVRAIT être utilisée comme guide pour choisir l'origine appropriée. Par exemple : si un nœud de configuration particulier, la déclaration associée YANG "description" indique qu'une valeur négociée par le protocole outrepasse une valeur configurée, l'origine sera alors rapportée comme "learned", même si la valeur apprise est la même que la valeur configurée. À l'inverse, si, pour un nœud de configuration particulier, la déclaration YANG "description" associée indique qu'une valeur négociée par le protocole n'outrepasse pas une valeur explicitement configurée, l'origine sera alors rapportée comme "intended", même si une valeur apprise est la même que celle configurée.

Dans le cas où un appareil ne peut pas fournir une origine précise pour un nœud particulier de données de configuration, il DEVRAIT utiliser l'origine "unknown".

6. Implications sur YANG

6.1 Contexte XPath

Ce paragraphe met à jour le paragraphe 6.4.1 of RFC 7950.

Si un serveur met en œuvre l'architecture définie dans le présent document, les arborescences accessibles pour certains contextes XPath sont précisées comme suit :

- Si l'expression XPath est définie dans une sous déclaration à un nœud de données qui représente l'état du système, l'arborescence accessible est tout état de fonctionnement dans le serveur. Le nœud racine a tous les nœuds de données de niveau supérieur dans tous les modules comme enfants.

- Si l'expression XPath est définie dans une sous déclaration à une déclaration "notification", l'arborescence accessible est l'instance de notification et tout état de fonctionnement dans le serveur. Si la notification est définie sur le niveau supérieur dans un module, alors le nœud racine a le nœud représentant la notification qui est définie et tous les nœuds de données de niveau supérieur dans tous les modules comme enfants. Autrement, le nœud racine a tous les nœuds de données de niveau supérieur dans tous les modules comme enfants.

- Si l'expression XPath est définie dans une sous déclaration comme une déclaration "input" dans une déclaration "rpc" ou "action", l'arborescence accessible est l'appel de procédure distante (RPC, *Remote Procedure Call*) ou l'instance de fonctionnement d'action et tout état de fonctionnement dans le serveur. Le nœud racine a des nœuds de données de niveau supérieur dans tous les modules comme enfants. De plus, pour un RPC, le nœud racine a aussi le nœud qui représente l'opération de RPC qui est en train d'être définie comme enfant. Le nœud qui représente l'opération en cours de définition a les paramètres d'entrée de l'opération comme enfants.

- Si l'expression XPath est définie dans une sous déclaration d'une déclaration "output" dans une déclaration "rpc" ou "action", l'arborescence accessible est le RPC ou l'instance d'opération d'action et tout état de fonctionnement dans le serveur. Le nœud racine a des nœuds de données de niveau supérieur dans tous les modules comme enfants. De plus, pour un RPC, le nœud racine a aussi le nœud qui représente l'opération de RPC en cours de définition comme enfant. Le nœud qui représente l'opération définie a les paramètres de résultat de l'opération comme enfants.

6.2 Invocation des actions et des RPC

Ce paragraphe met à jour le paragraphe 7.15 de la RFC 7950.

Les actions sont toujours invoquées dans le contexte du magasin de données d'état de fonctionnement. Le nœud pour lequel l'action est invoquée DOIT exister dans le magasin de données d'état de fonctionnement.

Noter que le présent document ne contraint en aucune façon le résultat de l'invocation d'un RPC ou d'une action. Par exemple, un RPC peut être défini pour modifier le contenu d'un magasin de données.

7. Modules YANG

<DÉBUT DE CODE> fichier "ietf-datastore@2018-02-14.yang"

```
module ietf-datastore {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-datastore";
    prefix ds;

organisation : "Groupe de travail IETF Network Modeling (NETMOD)";

contact  "WG Web:   < https://datatracker.ietf.org/wg/netmod/ >

Liste de diffusion du groupe de travail :  < mailto:netmod@ietf.org >

Auteur : Martin Bjorklund < mailto:mbj@tail-f.com >
Auteur : Juergen Schoenwaelder < mailto:j.schoenwaelder@jacobs-university.de >
Auteur : Phil Shafer < mailto:phil@juniper.net >
Auteur : Kent Watsen < mailto:kwatsen@juniper.net >
Auteur : Rob Wilton < rwilton@cisco.com >";

Description : "Ce module YANG définit un ensemble d'identités pour les magasins de données.

Copyright (c) 2018 IETF Trust et les personnes identifiées comme auteurs du code. Tous droits réservés.
La redistribution et l'utilisation en formes source et binaire, avec ou sans modification, est permise conformément et selon
les termes de la licence contenue dans la licence BSD simplifiée établie à la Section 4.c des dispositions du règlement
intérieur de l'IETF Trust relatives aux documents de l'IETF (https://trustee.ietf.org/license-info ).
Cette version de ce module YANG fait partie de la RFC 8342 (https://www.rfc-editor.org/info/rfc8342) ; voir dans la RFC
elle-même les notices légales complètes".

révision 2018-02-14 {
      description "Révision initiale.";
      référence "RFC 8342 : Architecture de magasin de données de gestion de réseau (NMDA)";
    }

/* * Identités */

identité magasin de données {
      description : "Identité de base abstraite pour les identités de magasin de données.";
}

identité conventionnelle {
     magasin de données de base ;
      description : "Identité de base abstraite pour configuration conventionnelle de magasin de données.";
}

identité courante {
      base conventionnelle;
      description : "Magasin de données de configuration en cours.";
}

identité candidate {
      base conventionnelle;
      description : "Magasin de données de configuration candidat.";
}
```

```
identité startup {
      base conventionnelle;
      description : "Magasin de données de configuration de démarrage.";
}

identité intended {
      base conventionnelle;
      description : "Magasin de données de configuration prévu.";
}

identité dynamic {
      base magasin de données;
      description : "Identité de base abstraite pour magasin de données de configuration dynamique.";
}

identité operational {
      base magasin de données;
      description : "Magasin de données d'état de fonctionnement.";
}

/*  * définitions de type */

typedef datastore-ref {
  type identityref {
    base magasin de données;
  }
  description : "Référence d'identité de magasin de données.";
}
}
```

<FIN DE CODE>

<DÉBUT DE CODE> fichier "ietf-origin@2018-02-14.yang"

```
module ietf-origin {
 yang-version 1.1;
 namespace "urn:ietf:params:xml:ns:yang:ietf-origin";
 prefix or;
 import ietf-yang-metadata {
  prefix md;
}

organisation : "Groupe de travail IETF Network Modeling (NETMOD)";

contact :
WG Web :  < https://datatracker.ietf.org/wg/netmod/ >
WG List :  < mailto:netmod@ietf.org >

Auteur :   Martin Bjorklund < mailto:mbj@tail-f.com >
Auteur :   Juergen Schoenwaelder < mailto:j.schoenwaelder@jacobs-university.de >
Auteur :   Phil Shafer  < mailto:phil@juniper.net >
Auteur :   Kent Watsen < mailto:kwatsen@juniper.net >
Auteur :   Rob Wilton < rwilton@cisco.com >;

description : "Ce module YANG définit une annotation de métadonnées 'origin' et un ensemble d'identités pour la valeur
d'origine.

Copyright (c) 2018 IETF Trust et les personnes identifiées comme auteurs du code. Tous droits réservés.
La redistribution et l'utilisation en formes source et binaire, avec ou sans modification, est permise conformément et selon
les termes de la licence contenue dans la licence BSD simplifiée établie à la Section 4.c des dispositions du règlement
intérieur de l'IETF Trust relatives aux documents de l'IETF (https://trustee.ietf.org/license-info ).
Cette version de ce module YANG fait partie de la RFC 8342 (https://www.rfc-editor.org/info/rfc8342) ; voir dans la RFC
```

```
elle-même les notices légales complètes".

révision 2018-02-14 {
description "Révision initiale.";
référence "RFC 8342 : Network Management Datastore Architecture (NMDA)";
 }

/* * Identités */

identité origin {
  description : "Identité de base abstraite pour l'annotation origin.";
}

identité intended {
  base origin;
  description : "Note la configuration à partir du magasin de données de configuration prévu.";
}

identité dynamic {
  base origin;
  description  : "Note la configuration à partir d'un magasin de données de configuration dynamique.";
}

identité system {
  base origin;
   description : "Note la configuration générée par le système lui-même. Des exemples de configuration système incluent la
configuration appliquée pour une interface de bouclage toujours existante, ou une configuration d'interface qui est auto-
créée du fait du matériel actuellement présent dans l'appareil.";
}

identité learned {
  base origin;
  description : "Note la configuration apprise des interactions de protocole avec d'autres appareils, au lieu de via le magasin
de données de configuration prévu ou un magasin de données de configuration dynamique. Des exemples de protocoles qui
fournissent la configuration apprise incluent des négociations de couche de liaison, des protocoles d'acheminement, et
DHCP.";
}

identité default {
  base origin;
   description : "Note une configuration qui n'a pas une valeur configurée ou apprise mais utilise une valeur par défaut.
Couvre à la fois les valeurs définies dans une déclaration 'default' et les valeurs définies via une explication dans une
déclaration 'description'.";
}

identité unknown {
  base origin;
  description : "Note une configuration dont le système ne peut pas identifier l'origine.";
}

/* * Définitions de type */

typedef origin-ref {
 type identityref {
   base origin;
 }
 description "Référence d'identité d'origine.";
}

/* * Annotations de métadonnées */

md:annotation origin {
 type origin-ref;
```

```
    description : "L'annotation 'origin' peut être présente dans tout nœud de données de configuration dans le magasin de
données d'état de fonctionnement. Elle spécifie d'où le nœud est originaire. Si elle n'est pas spécifiée pour un certain nœud
de données de configuration, l'origine est alors la même que celle de son nœud parent dans l'arborescence des données.
L'origine pour tous les nœuds de données de configuration de niveau supérieur doit être spécifiée.";
  }
}
```

<FIN DU CODE>

8. Considérations relatives à l'IANA

8.1 Mise à jour du registre XML de l'IETF

Le présent document enregistre deux URI dans le registre XML de l'IETF [RFC3688]. Suivant le format de la [RFC3688], les enregistrements suivants ont été faits :

URI : urn:ietf:params:xml:ns:yang:ietf-datastore
Contact d'enregistrement : IESG.
XML : non disponible ; l'URI demandé est un espace de noms XML.

URI : urn:ietf:params:xml:ns:yang:ietf-origin
Contact d'enregistrement : IESG.
XML : non disponible ; l'URI demandé est un espace de noms XML.

8.2 Mise à jour du registre des noms de module YANG

Le présent document enregistre deux modules YANG dans le registre "Noms de modules YANG" [RFC6020]. Suivant le format de la [RFC6020], les enregistrements suivants ont été faits :

nom : ietf-datastore
espace de noms : urn:ietf:params:xml:ns:yang:ietf-datastore
préfixe : ds
référence : RFC 8342

nom : ietf-origin
espace de noms : urn:ietf:params:xml:ns:yang:ietf-origin
préfixe : or
référence : RFC 8342

9. Considérations sur la sécurité

Le présent document discute d'un modèle architectural de magasin de données pour la gestion de réseau utilisant NETCONF/RESTCONF et YANG. Il n'a pas d'impact sur la sécurité dans l'Internet.

Bien que ce document spécifie plusieurs modules YANG, ces modules définissent seulement les identités et une annotation de métadonnées ; donc, les "directives pour la sécurité du module YANG" [YANG-SEC] ne s'appliquent pas.

L'annotation de métadonnées d'origine expose l'origine des valeurs dans la configuration appliquée. Les informations d'origine peuvent fournir l'indication que certains protocoles de plan de contrôle sont actifs sur l'appareil. Comme les informations d'origine sont liées aux valeurs de configuration appliquée, elles ne sont accessibles qu'aux clients qui ont les permissions de lire les valeurs de configuration appliquée. Les administrateurs de sécurité devraient considérer la sensibilité des informations d'origine lorsque ils définissent les règles de contrôle d'accès.

10. Références

10.1 Références normatives

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174). DOI 10.17487/RFC2119.

[RFC6241] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, "Protocole de configuration de réseau (NETCONF)", juin 2011. *(Remplace la* RFC4741) *(P.S. ; MàJ par* RFC7803, RFC8526), DOI 10.17487/RFC6241.

[RFC7950] M. Bjorklund, "Langage de modélisation de données YANG 1.1", août 2016. *(P.S. ; MàJ par* RFC8526*)*, DOI 10.17487/RFC7950.

[RFC7952] L. Lhotka, "Définition et utilisation de métadonnées avec YANG", août 2016. *(P.S.)*, DOI 10.17487/RFC7952.

[RFC8040] A. Bierman, et autres, "Protocole RESTCONF", janvier 2017. *(P.S. ; MàJ par* RFC8527*)*, DOI 10.17487/RFC8040.

[RFC8174] B. Leiba, "Ambiguïté des mots clés en majuscules ou minucules dans la RFC2119", mai 2017. BCP14. *(MàJ* RFC2119), DOI 10.17487/RFC8174.

[XML-2008] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, novembre 2008, <https://www.w3.org/TR/2008/REC-xml-20081126 >.

10.2 Références pour information

[NETMOD-Operational] Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", Travail en cours, draft-bjorklund-netmod-operational-00, octobre 2012.

[OpState-Enhance] Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", Travail en cours, draft-kwatsen-netmod-opstate-02, février 2016.

[OpState-Modeling] Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", Travail en cours, draft-openconfig-netmod-opstate-01, juillet 2015.

[OpState-Reqs] Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", Travail en cours, draft-ietf-netmod-opstate-reqs-04, janvier 2016.

[RFC3688] M. Mealling, "Registre XML de l'IETF", BCP 81, janvier 2004. DOI 10.17487/RFC3688.

[RFC6020] M. Bjorklund, "YANG - Langage de modélisation des données pour le protocole de configuration de réseau (NETCONF)", octobre 2010. *P.S.)*, DOI 10.17487/RFC6020.

[RFC6244] P. Shafer, "Architecture pour la gestion de réseau avec NETCONF et YANG", juin 2011. *(Information)*, DOI 10.17487/RFC6244.

[RFC8343] M. Bjorklund, "Modèle de données YANG pour la gestion d'interface", mars 2018. *(P.S., remplace* RFC7223), DOI 10.17487/RFC8343.

[RFC8344] M. Bjorklund, "Modèle de données YANG pour la gestion IP", mars 2018. *(P.S., remplace* RFC7277), DOI 10.17487/RFC8344.

[With-config-state] Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", Travail en cours, draft-wilton-netmod-opstate-yang-02, décembre 2015.

[YANG-SEC] IETF, "Directives sur la sécurité de YANG", < https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines >.

Appendice A. Directives sur la définition des magasins de données

La définition d'un nouveau magasin de données dans cette architecture devrait être fournie dans un document (par exemple, une RFC) destiné à définir le magasin de données. Lorsque c'est justifié, plusieurs magasins de données peuvent être définis dans le même document (par exemple, lorsque les magasins de données sont logiquement connectés). Chaque définition de magasin de données devrait traiter les points spécifiés dans les paragraphes ci-dessous.

A.1 Définir quels modules YANG peuvent être utilisés dans le magasin de données

Tous les modules YANG ne peuvent pas être utilisés dans tous les magasins de données. Certains magasins de données peuvent imposer des contraintes aux modèles de données qui peuvent être utilisés en leur sein. Si il est souhaitable qu'un sous ensemble des modules soit ciblé pour le magasin de données, la documentation qui définit le magasin de données doit l'indiquer.

A.2 Définir quel sous ensemble de données modélisées de YANG s'applique

Par défaut, les données dans un magasin de données sont modélisées par toutes les déclarations YANG dans les modules YANG disponibles. Cependant, il est possible de spécifier des critères que doivent satisfaire les déclarations YANG afin d'être présentes dans un magasin de données. Par exemple, peut-être que seulement les nœuds "config true", ou les nœuds "config false" qui ont aussi une extension YANG spécifique, sont présents dans le magasin de données.

A.3 Définir comment les données sont actualisées

Le nouveau magasin de données doit spécifier comment il interagit avec les autres magasins de données.

Par exemple, le diagramme de la Section 5 décrit l'alimentation d'un magasin de données de configuration dynamique dans <operational>. Comment cette interaction se produit doit être défini par le magasin de données de configuration dynamique particulier. Dans certains cas, elle peut se produire implicitement aussitôt que les données sont mises dans le magasin de données de configuration dynamique, tandis que dans d'autres cas, une action explicite (par exemple, un RPC) peut être exigé pour déclancher l'application des données du magasin de données.

A.4 Définir quels protocoles peuvent être utilisés

Par défaut, on suppose que les deux protocoles NETCONF et RESTCONF peuvent être utilisés pour interagir avec un magasin de données. Cependant, il se peut que seul un protocole spécifique puisse être utilisé (par exemple, le protocole de séparation d'élément de transmission et de contrôle (ForCES, *Forwarding and Control Element Separation*)) ou qu'un sous ensemble de toutes les opérations ou capacités de protocole soit disponible (par exemple, pas de verrouillage ou pas de filtrage fondé sur XPath).

A.5 Définir les identités YANG pour le magasin de données

Le magasin de données doit être défini avec une identité YANG qui utilise l'identité "ds:datastore", ou une de ses identités dérivées, comme base. Cette identité est nécessaire, afin que le magasin de données puisse être référencé dans les opérations de protocole (par exemple, <get-data>).

Le magasin de données peut aussi être défini par une identité qui utilise l'identité "or:origin", ou une de ses identités dérivées, comme base. Cette identité est nécessaire si le magasin de données interagit avec <operational>, afin que les données originaires du magasin de données puissent être identifiées comme telles via l'attribut de métadonnées "origin" défini à la Section 7.

Un exemple de ces lignes directrices figure dans l'Appendice B.

Appendice B. Exemple d'un magasin de données de configuration dynamique éphémère

Cette Section définit la documentation d'un exemple d'un magasin de données de configuration dynamique utilisant les lignes directrices fournies dans l'Appendice A. En bref, seul un exemple réduit est montré ; on suppose qu'une RFC autonome sera écrite lorsque ce type de scénario sera complètement pris en compte.

Cet exemple définit un magasin de données de configuration dynamique appelé "éphémère", qui est en gros modélisé

d'après les travaux du groupe de travail I2RS.

Nom	Valeur
Nom	éphémère
Modules YANG	tous (par défaut)
Nœuds YANG	tous les nœuds de données "config true"
Comment l'appliquer	changements automatiquement propagés à <operational>
Protocoles	NETCONF/RESTCONF (par défaut)
Module YANG de définition	"exemple-ds-éphémère"

Propriétés de l'exemple de magasin de données "éphémère"

```
module exemple-ds-éphémère {
  yang-version 1.1;
  namespace "urn:exemple-ds-éphémère";
 prefix eph;

import ietf-datastore {
 prefix ds;
}

import ietf-origin {
 prefix or;
}

// identité magasin de données
 identité ds-ephemeral {
   base ds:dynamic;
   description : "Le magasin de données de configuration dynamique éphémère.";
 }

// identité origin
  identité or-ephemeral {
   base or:dynamic;
   description : "Note les données provenant du magasin de données de configuration dynamique éphémère.";
 }
}
```

Appendice C. Exemple de données

L'utilisation des magasins de données est complexe, et beaucoup des effets subtils sont plus facilement présentés en utilisant des exemples. Le présent appendice présente une série d'exemples de modèles de données avec un échantillon de contenu de divers magasins de données.

Les fragments XML [XML-2008] qui suivent ne sont fournis que comme exemples.

C.1 Exemple de système

Dans cet exemple, le module fictif suivant est utilisé :

```
  module example-system {
   yang-version 1.1;
   namespace urn:example:system;
   prefix sys;

   import ietf-inet-types {
     prefix inet;
   }

   container system {
     leaf hostname {
       type string;
```

```
    }

    list interface {
     key name;

     leaf name {
      type string;
     }

     container auto-negotiation {
      leaf enabled {
       type boolean;
       default true;
      }
      leaf speed {
       type uint32;
       units mbps;
       description : "Vitesse annoncée en Mbps.";
      }
     }

     leaf speed {
      type uint32;
      units mbps;
      config false;
      description :"Vitesse de l'interface, en Mbps.";
     }

     list address {
      key ip;
      leaf ip {
       type inet:ip-address;
      }
      leaf prefix-length {
       type uint8;
      }
     }
    }
   }
  }
```

L'opérateur a configuré le nom d'hôte et deux interfaces, de sorte que le contenu de <intended> est :

```
<system xmlns="urn:example:system">

  <hostname>foo.example.com</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

  <interface>
    <name>eth1</name>
    <address>
      <ip>2001:db8::20</ip>
      <prefix-length>64</prefix-length>
```

```
      </address>
  </interface>

</system>
```

Le système a détecté que le matériel pour une des interfaces configurées ("eth1") n'est pas encore présent, de sorte que la configuration pour cette interface n'est pas appliquée. De plus, le système a reçu un nom d'hôte et une adresse IP supplémentaire pour "eth0" sur DHCP. En plus de remplir la valeur par défaut pour la feuille à capacité d'auto négociation, une entrée d'interface de bouclage est aussi automatiquement instanciée par le système. Tout cela est reflété dans <operational>. Noter comment l'attribut de métadonnées "origin" pour plusieurs nœuds de données "config true" est hérité de ses nœuds de données parents.

```
<system
    xmlns="urn:example:system"
    xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

   <hostname or:origin="or:learned">bar.example.com</hostname>

  <interface or:origin="or:intended">
    <name>eth0</name>
    <auto-negotiation>
      <enabled or:origin="or:default">true</enabled>
      <speed>1000</speed>
    </auto-negotiation>
    <speed>100</speed>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>64</prefix-length>
    </address>
    <address or:origin="or:learned">
      <ip>2001:db8::1:100</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

  <interface or:origin="or:system">
    <name>lo0</name>
    <address>
      <ip>::1</ip>
      <prefix-length>128</prefix-length>
    </address>
  </interface>

</system>
```

C.2 Exemple dans BGP

Considérons le fragment suivant d'un module BGP fictif :

```
 container bgp {
   leaf local-as {
     type uint32;
   }
   leaf peer-as {
     type uint32;
   }
   list peer {
     key name;
     leaf name {
       type inet:ip-address;
     }
     leaf local-as {
```

```
      type uint32;
      description : "... Par défaut pour ../local-as.";
    }
    leaf peer-as {
      type uint32;
      description : "... Par défaut pour ../peer-as.";
     }
      leaf local-port {
      type inet:port;
    }
    leaf remote-port {
     type inet:port;
     default 179;
    }
    leaf state {
      config false;
      type enumeration {
        enum init;
        enum established;
        enum closing;
      }
    }
  }
}
```

Dans cet exemple, bgp/peer/local-as et bgp/peer/peer-as ont tous deux des valeurs hiérarchiques complexes, permettant à l'utilisateur de spécifier des valeurs par défaut pour tous les homologues dans un seul lieu.

Le modèle suit aussi le schéma d'intégration complète de nœuds d'état ("config false") avec des nœuds de configuration ("config true"). Il n'y a pas de hiérarchie séparée "bgp-state", avec la répétition qui l'accompagne de nœuds de contenu et de désignation. Cela rend le modèle plus simple et plus lisible.

C.2.1 Magasins de données

Chaque magasin de données représente différentes vues de ces nœuds. <running> va contenir la configuration fournie par l'opérateur -- par exemple, un seul homologue BGP. <intended> va contenir conceptuellement les données telles que validées, après la suppression des données non destinées à la validation et après avoir effectué tous les mécanismes de gabarit local. <operational> va montrer les données provenant de <intended> ainsi que de tout nœud "config false".

C.2.2 Ajout d'un homologue

Si l'utilisateur configure un seul homologue BGP, cet homologue sera alors visible dans <running> et dans <intended>. Il peut aussi apparaître dans <candidate> si le serveur prend en charge le magasin de données de configuration candidat. La restitution de l'homologue va seulement retourner les valeurs spécifiées par l'utilisateur.

Aucun délai ne devrait exister entre l'apparition de l'homologue dans <running> et dans <intended>.

Dans ce scénario, on a ajouté ce qui suit à <running> :

```
<bgp>
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
  </peer>
</bgp>
```

C.2.2.1 <operational>

Le magasin de données "operational" va contenir les données d'homologue complètement étendues, incluant les nœuds "config false". Dans notre exemple, cela signifie que le nœud "state" va apparaître.

De plus, <operational> va contenir les valeurs "actuellement utilisées" pour tous les nœuds. Cela signifie que local-as et peer-as seront remplis même si ce ne sont pas des valeurs données dans <intended>. La valeur de bgp/local-as sera utilisée si bgp/peer/local-as n'est pas fournie ; bgp/peer-as et bgp/peer/peer-as auront la même relation. Du point de vue de "operational", cela signifie que chaque homologue aura des valeurs pour son local-as et peer-as, même si ces valeurs ne sont pas explicitement configurées mais sont fournies par bgp/local-as et bgp/peer-as.

Chaque homologue BGP a une connexion TCP associée, en utilisant les valeurs de local-port et remote-port provenant de <intended>. Si ces valeurs ne sont pas fournies, le système va choisir des valeurs. Lorsque la connexion est établie, <operational> va contenir les valeurs courantes pour les nœuds local-port et remote-port sans considération de l'origine. Si le système a choisi les valeurs, l'attribut "origin" sera réglé à "system". Avant que la connexion soit établie, un des nœuds, ou les deux peuvent ne pas apparaître, car le système peut ne pas avoir encore leurs valeurs.

```
<bgp xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
     or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>established</state>
  </peer>
 </bgp>
```

C.2.3 Suppression d'un homologue

Les changements de configuration peuvent prendre du temps pour percoler à travers les divers composants logiciels impliqués. Durant cette période, il est impératif de continuer à donner une vue précise du travail de l'appareil. <operational> va contenir des nœuds pour la configuration précédente et actuelle, aussi proche que possible du traçage du fonctionnement réel de l'appareil.

Considérons le scénario où on supprime un homologue BGP. Lorsque un homologue est supprimé, l'état de fonctionnement va continuer à refléter l'existence de cet homologue jusqu'à ce que les ressources de l'homologue soient libérées, incluant de clore la connexion de l'homologue. Durant cette période, les valeurs des données en cours vont continuer d'être visibles dans <operational>, avec l'attribut "origin" réglé à indiquer l'origine des données originales.

```
<bgp xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
    or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>closing</state>
  </peer>
</bgp>
```

Une fois que les ressources sont libérées et que la connexion est close, les données de l'homologue sont supprimées de <operational>.

C.3 Exemple d'interface

Dans ce paragraphe, on utilise ce simple modèle de données d'interface :

```
container interfaces {
 list interface {
   key name;
```

```
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
    }
  }
}
```

C.3.1 Interfaces préprovisionnées

Une problème courant dans les appareils de réseautage est la prise en charge des unités de champ remplaçables (FRU, *Field Replaceable Unit*) qui peuvent être insérées et retirées de l'appareil sans exiger un réamorçage ou d'interférer avec le fonctionnement normal. Ces FRU sont normalement des cartes d'interface, et les appareils prennent en charge le pré provisionnement de ces interfaces.

Si un client crée une interface "et-0/0/0" mais si l'interface n'existe pas physiquement à ce moment, <intended> peut alors contenir ce qui suit :

```
<interfaces>
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
  </interface>
</interfaces>
```

Comme l'interface n'existe pas, ces données n'apparaissent pas dans <operational>.

Lorsque une FRU contenant cette interface est insérée, le système va la détecter et traiter la configuration associée. <operational> va contenir les données provenant de <intended>, ainsi que les nœuds ajoutés par le système, tels que la valeur courante de la MTU de l'interface.

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
          or:origin="or:intended">
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
    <mtu or:origin="or:system">1500</mtu>
  </interface>
</interfaces>
```

Si la FRU est retirée, les données de l'interface sont retirées de <operational>.

C.3.2 Interface fournie par le système

Imaginons que le système fournisse une interface de bouclage (nommée "lo0") avec une adresse IPv4 par défaut de "127.0.0.1" et une adresse IPv6 par défaut de "::1". Le système ne va fournir une configuration pour cette interface que si il n'y a pas de données dans <intended>.

Lorsque aucune configuration pour "lo0" n'apparaît dans <intended>, <operational> va montrer les données fournies par le système:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
          or:origin="or:intended">
  <interface or:origin="or:system">
    <name>lo0</name>
```

```
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Lorsque la configuration pour "lo0" apparaît dans <intended>, <operational> va montrer les données dont l'origine est réglée à "intended". Si la "ip-address" n'est pas fournie, la valeur fournie par le système va alors apparaître comme suit :

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
        or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Remerciements

Le présent document est sorti de nombreuses discussions qui ont eu lieu depuis 2010. Plusieurs documents ([NETMOD-Operational] [With-config-state] [OpState-Reqs] [OpState-Enhance] [OpState-Modeling], ainsi que la [RFC6244]) ont abordé certains des problèmes du modèle d'origine du magasin de données. Les personnes suivantes sont les auteurs de ces travaux en cours ou ont été par ailleurs activement impliquées dans les discussions qui ont conduit à ces document :

- Lou Berger, LabN Consulting, L.L.C., < lberger@labn.net >
- Andy Bierman, YumaWorks, < andy@yumaworks.com >
- Marcus Hines, Google, < hines@google.com >
- Christian Hopps, Deutsche Telekom, < chopps@chopps.org >
- Balazs Lengyel, Ericsson, < balazs.lengyel@ericsson.com >
- Ladislav Lhotka, CZ.NIC, < lhotka@nic.cz >
- Acee Lindem, Cisco Systems, < acee@cisco.com >
- Thomas Nadeau, Brocade Networks, < tnadeau@lucidvision.com >
- Tom Petch, Engineering Networks Ltd, < ietfc@btconnect.com >
- Anees Shaikh, Google, < aashaikh@google.com >
- Rob Shakir, Google, < robjs@google.com >
- Jason Sterne, Nokia, < jason.sterne@nokia.com >

Juergen Schoenwaelder a été partiellement financé par Flamingo, un projet de réseau d'excellence (ICT-318488) soutenu par la Commission Européenne dans son septième programme cadre.

Adresse des auteurs

Martin Bjorklund
Tail-f Systems
mél : mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University
mél : j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks
mél : phil@juniper.net

Kent Watsen
Juniper Networks
mél : kwatsen@juniper.net

Robert Wilton
Cisco Systems
mél : rwilton@cisco.com