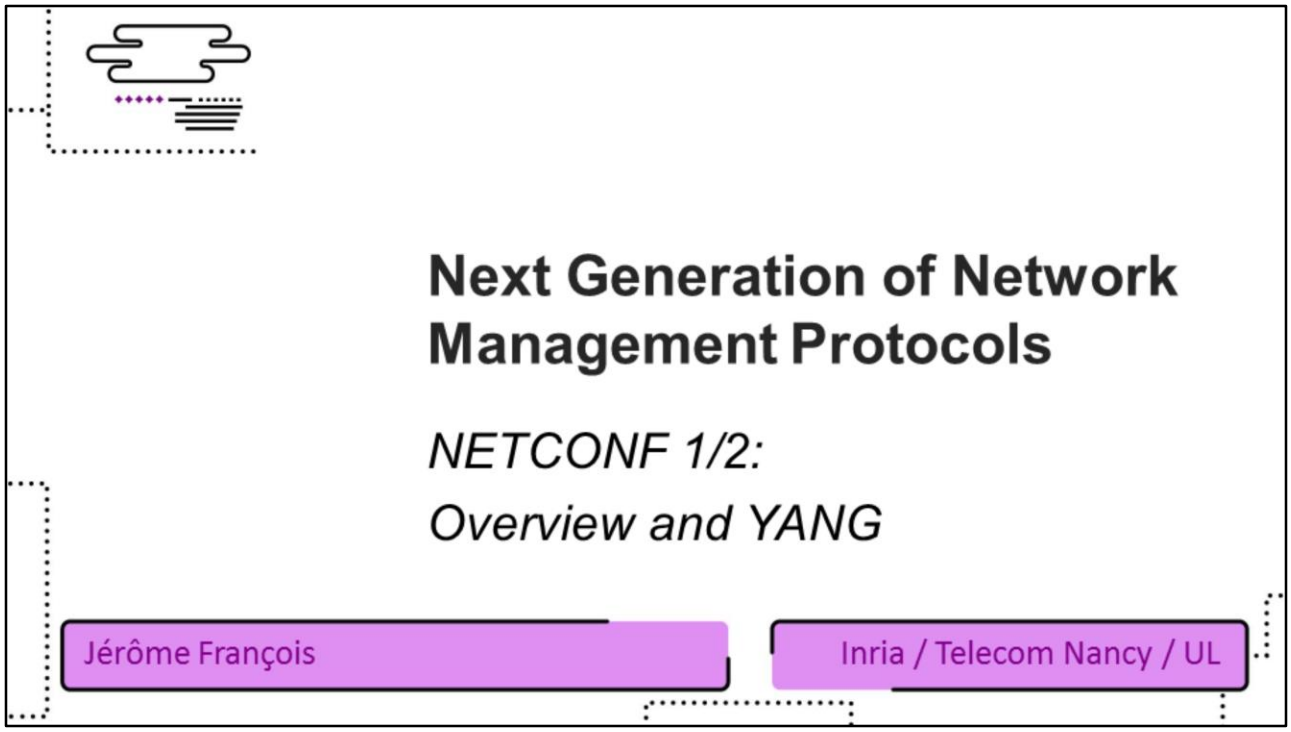
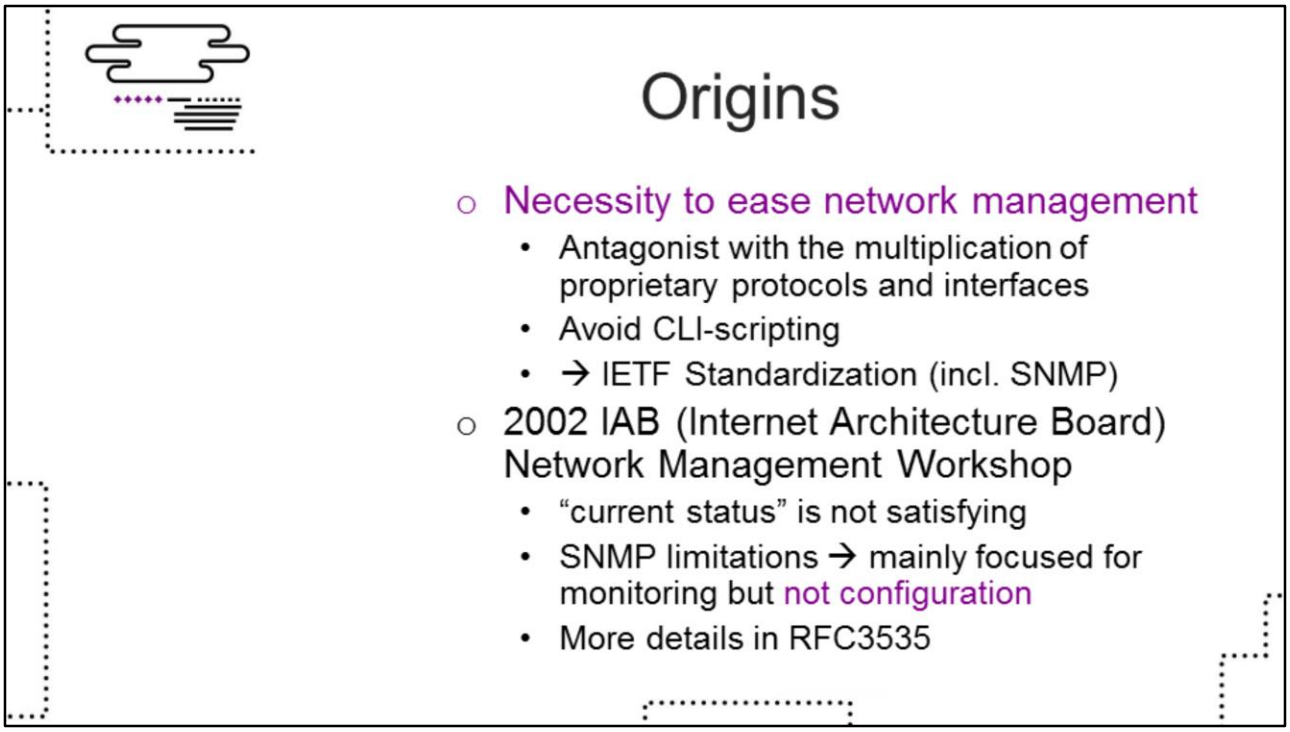


Bonjour à tous, nous allons aborder dans ce cours, le protocole de gestion de configuration NETCONF.

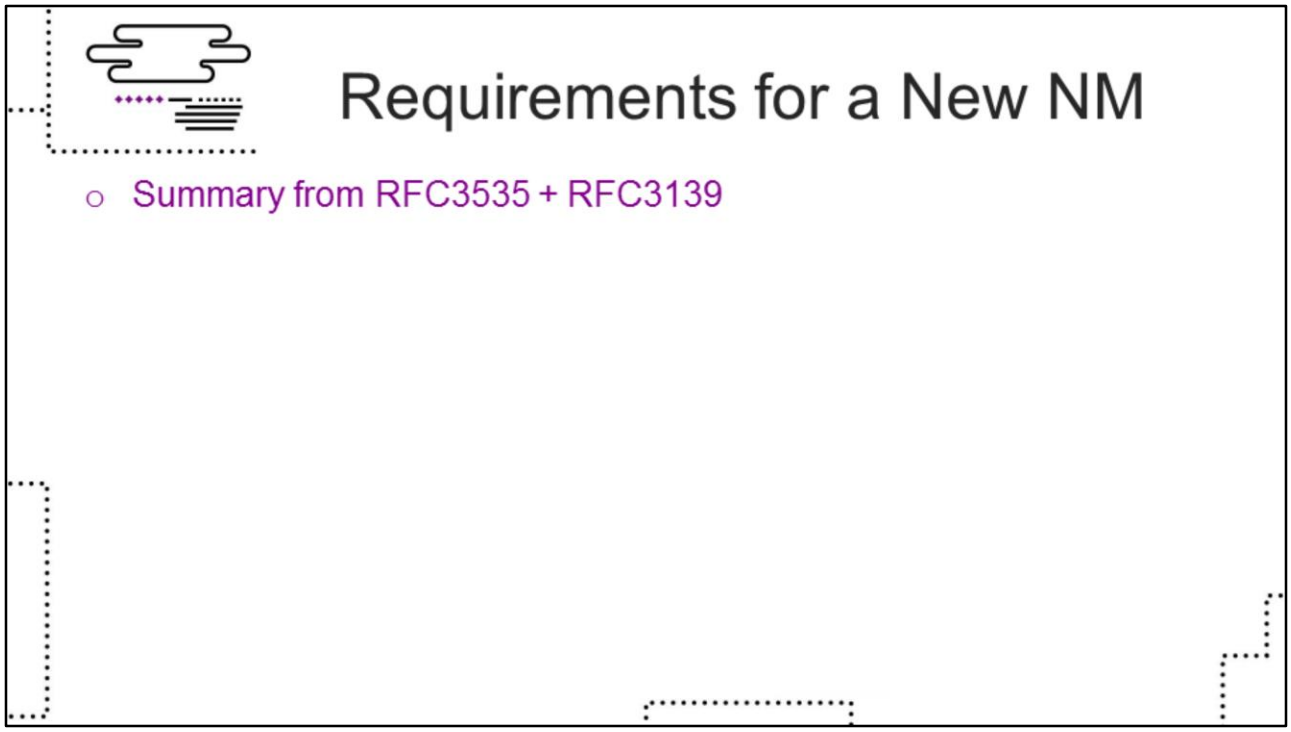


Pour commencer, quelques mots sur les origines de NETCONF.
Celles-ci remontent au début des années 2000. En effet, à ce moment, les opérateurs réseaux ont été confrontés à la multiplication des équipements et services à superviser. avec beaucoup d'interfaces propriétaires distinctes. La supervision s'en retrouverait donc très difficile et était largement basée sur du scripting à la main difficilement maintenable. Ce genre de méthodes n'était donc plus du tout adapté à l'Internet et posait de sérieux problèmes de passage à l'échelle. Plus d'automatisation et d'interfaces normalisées étaient nécessaires.

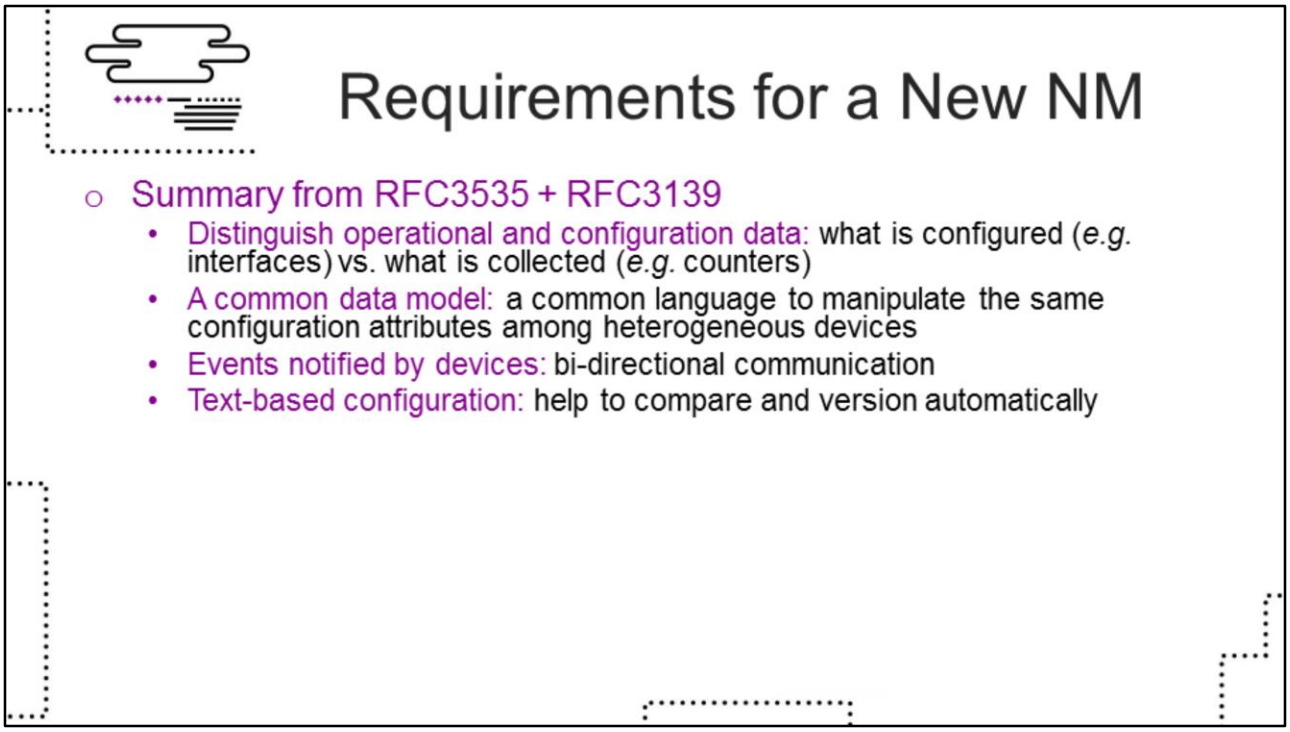
Il est vrai cependant que des efforts avaient déjà été fourni auparavant et notamment avec SNMP. Au début des années 2000, il a été jugé lors d'une réunion à l'IETF ,à l'Internet Architecture Board, que les technologies et protocoles de gestion n'avaient pas atteint un niveau satisfaisant.
Pour info ou rappel, l'IETF est un des organismes de standardisation majeur pour les protocoles réseaux.

Je vous invite particulièrement à consulter le RFC3535 qui récapitule les discussions de cette réunion et tiens à vous citer quelques points marquants, notamment par rapport à SNMP. D'une part, SNMP de son fonctionnement simple a largement été adopté avec la définition de MIB (Management Information bBse) pour beaucoup d'équipements et services. Cependant, de nombreuses d'entre elles étaient propriétaires malgré certains efforts de standardisation. L'expérience démontre que la

passage à l'échelle de SNMP est problématique. De plus, SNMP est principalement utilisé pour des fonctions de monitoring plus que pour de la configuration, la plupart des MIBs n'étant pas prévue pour ce cas là. SNMP est un protocole simple qui, de manière très caricaturale, sert donc uniquement à transférer des données de configuration alors que la gestion réseau consiste également à appliquer des procédures de configurations pouvant être complexes.

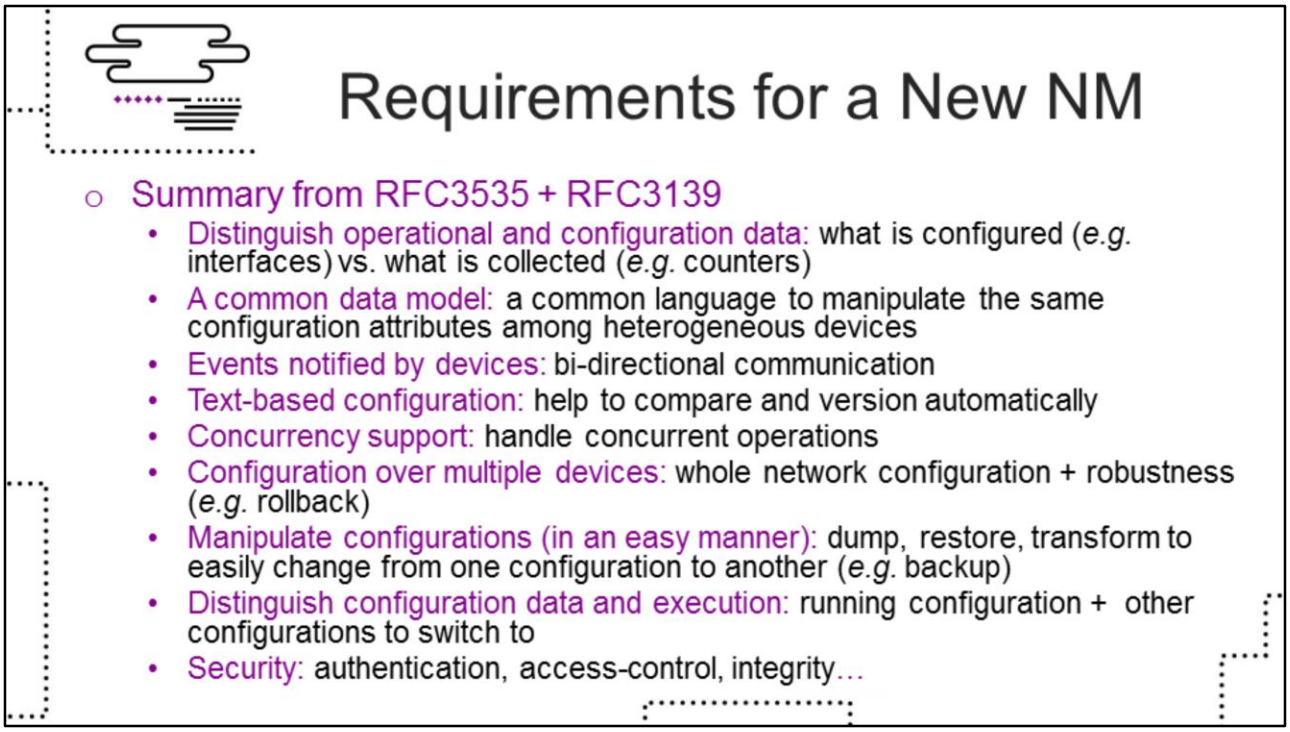


Face à ces observations, un certain nombre de besoins ont été listés et les principaux sont résumés ici.



Tout d'abord, il faut faire une distinction nette entre ce qui est configurable (par exemple les interfaces réseaux d'un routeur) et les données de monitoring, type statistiques (par exemple un compteur sur une interface).
Deuxièmement, il faut une interface et un langage standard pour manipuler les mêmes éléments de configuration, et ce même si les équipements ou les fabricants diffèrent. Un exemple simple est une adresse IP que l'on peut configurer pour beaucoup d'équipements et qui devrait donc être décrite de la même manière partout.

Un autre besoin est la nécessité d'avoir une communication bidirectionnelle notamment pour des remontée d'évènements automatiques
Egalement, Un protocole en mode texte pour faire de la comparaison de versions. En effet, comme pour le développement logiciel où les systèmes de versioning facilitent le développement collaboratif et la maintenance, les configurations réseaux étant de plus en plus complexes on attend le même type de mécanismes.

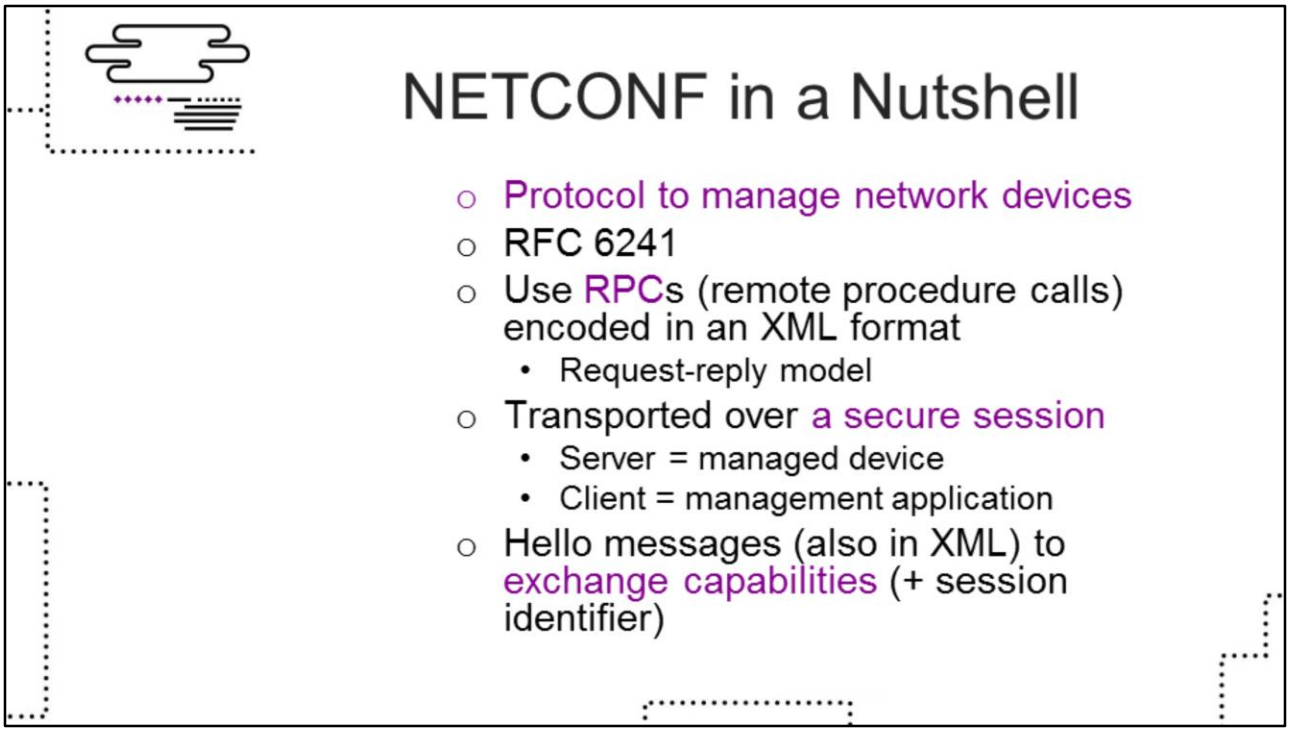


Et comme c'est plus complexe, on veut pouvoir aussi supporter des opérations de configurations en parallèle et non plus seulement configurer un à un des équipements ou automatiser cela avec un script à la main, souvent fastidieux.

Dans la même veine, il est également souhaitable que l'on puisse facilement installer ou changer une configuration complète. Et directement liée à cela, la juxtaposition de plusieurs configurations avec celle en cours d'exécution est souhaitée, notamment pour en préparer une en arrière plan et la mettre en place une fois que tout est prêt.

A cela s'ajoutent des propriétés attendues de sécurité inévitables aujourd'hui.

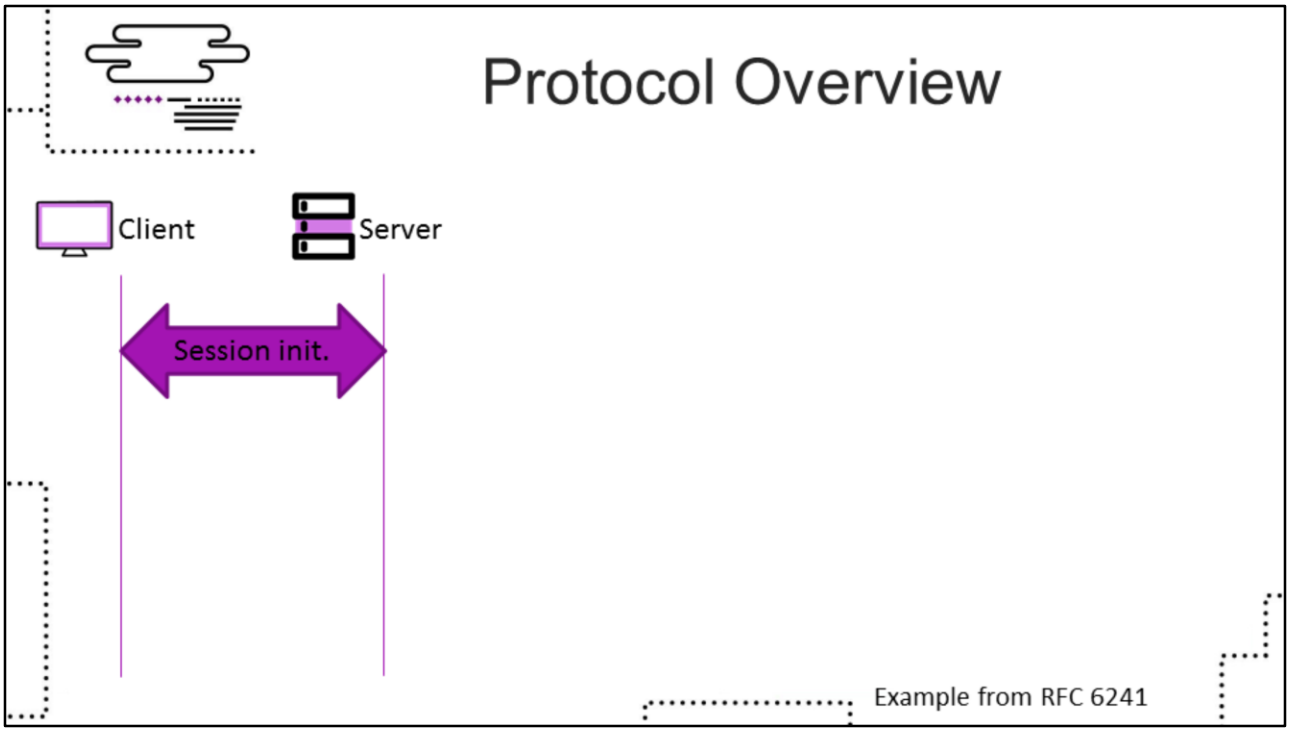
Nous voilà donc avec une liste de besoins exprimés dans lesquels NETCONF trouvent ses origines et tend donc à y répondre.



De manière très synthétique, NETCONF est un protocole de gestion ou supervision des réseaux qui permet de manipuler les différentes données de configuration sur un équipement réseau (physique ou virtuel d'ailleurs). Il fait l'objet de nombreux RFCs dont le 6241.

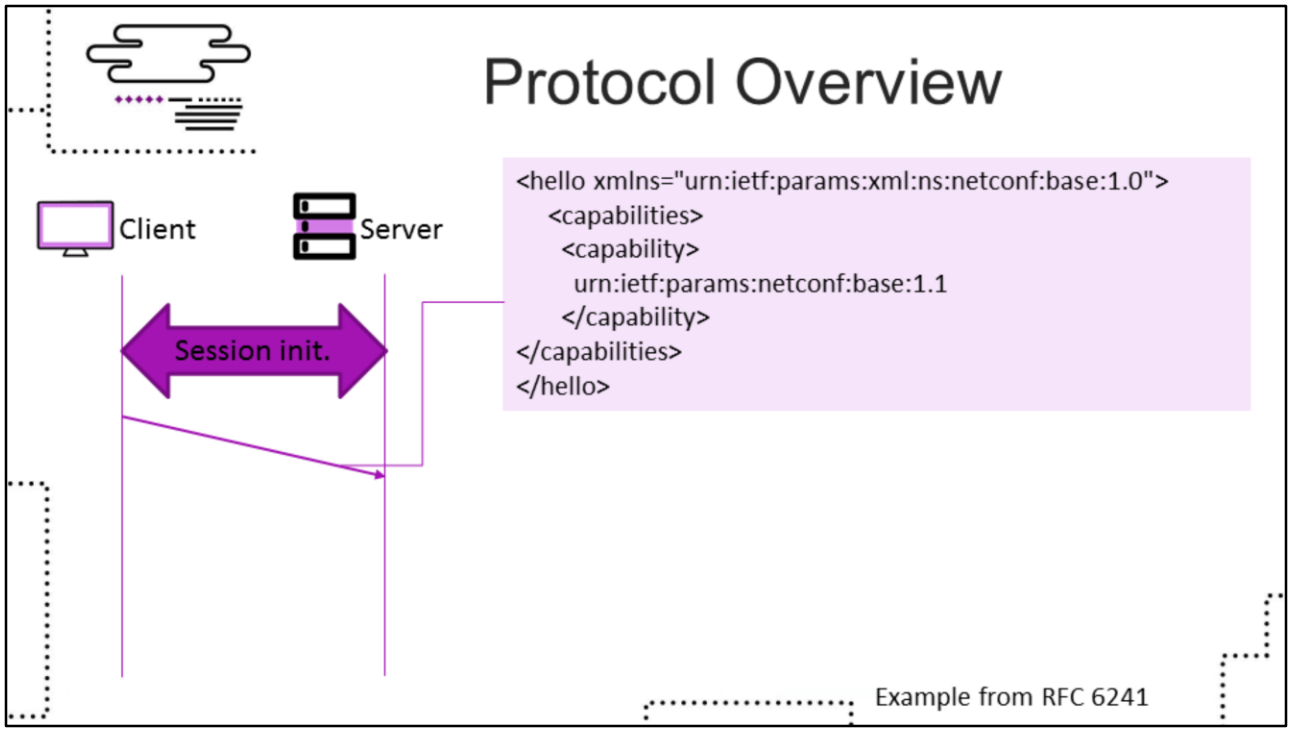
Le but n'était bien entendu pas de repartir de zéro mais de s'appuyer sur des protocoles existants. NETCONF s'appuit donc sur les RPCs (Remote Procedure Calls) pour exécuter des traitements de configuration distants. Ces appels et réponses sont encodés sous un format XML selon un schema bien défini et selon un modèle requête-réponse. Afin de garantir les propriétés de sécurité, ces appels RPCs sont transmis par le biais d'un canal sécurisé principalement un canal SSH.

Une fois initialisée par le protocole sous-jacent, une session NETCONF commence avec l'échange de messages HELLO. Le but principal de cet échange initial est de connaître les capacités de l'équipement supervisé, c'est à dire ce qu'il est possible de faire en termes de configuration. Un identifiant de session est également choisi par le serveur

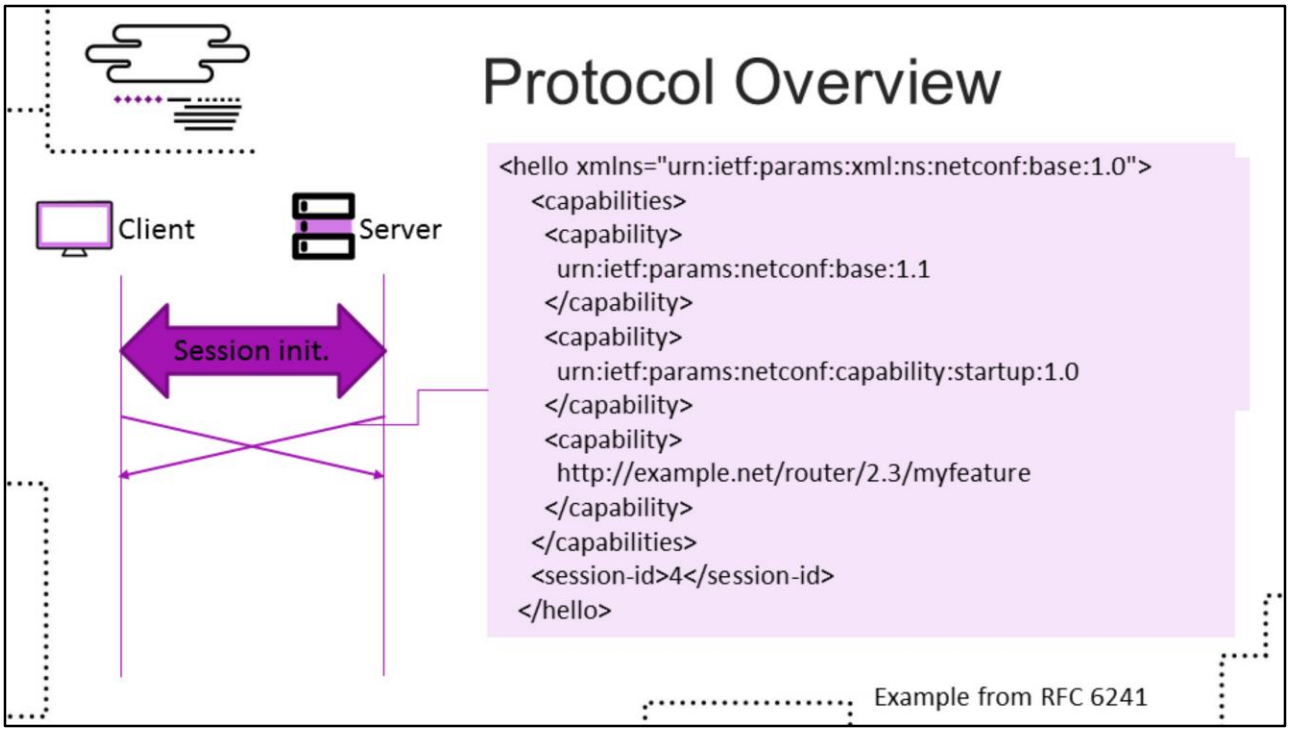


Voilà en détail ce que l'on peut observer

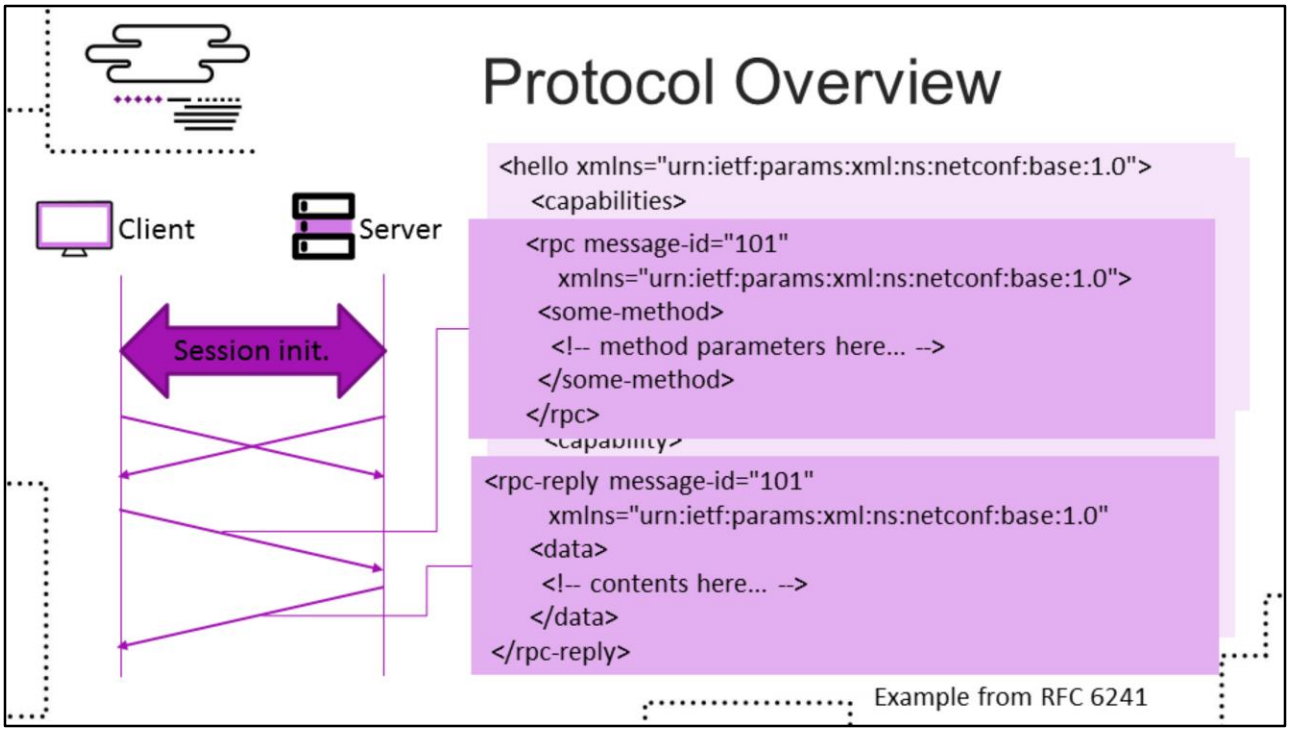
La session est tout d'abord initialisée entre le client et le serveur. Le serveur se trouve donc sur l'équipement supervisé.



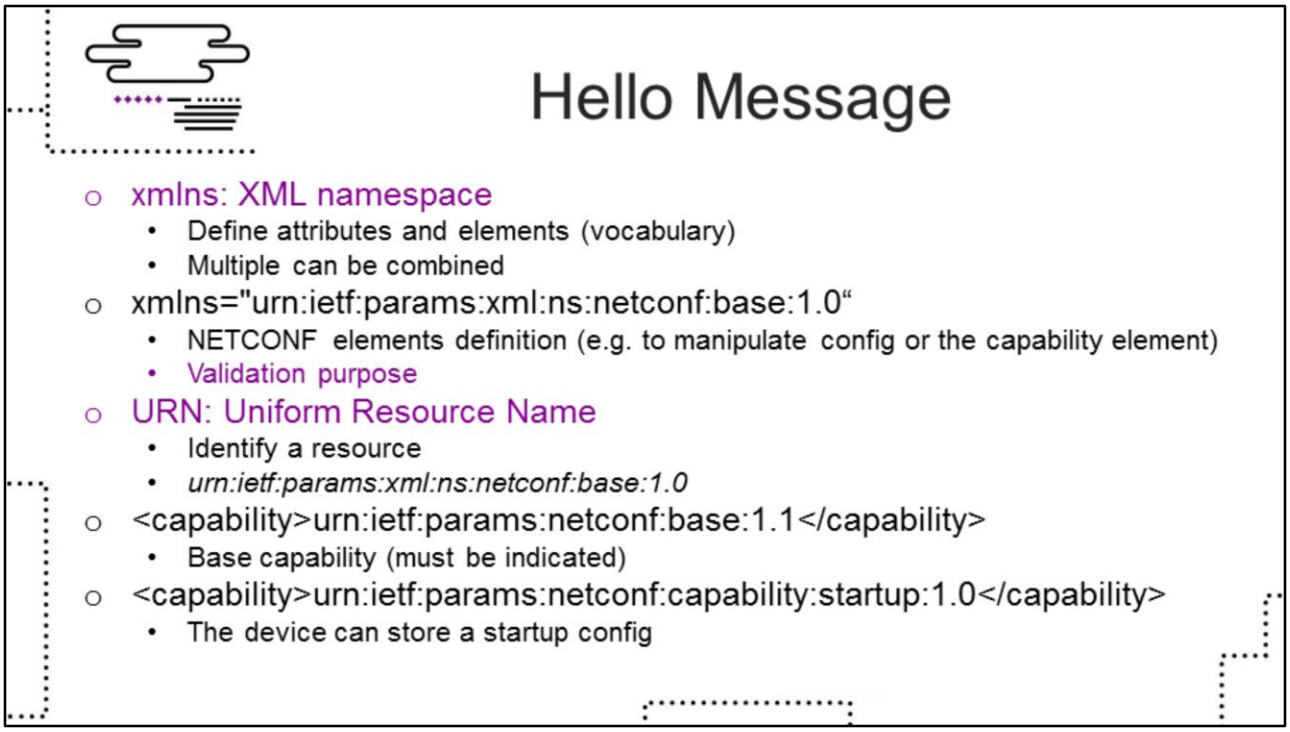
Le client envoie alors ses capacités car même celui-ci doit le faire et au minimum annoncer les capacités dites de bases ici en version 1.1 (version depuis Juin 2011). Il est possible également de supporter une version antérieur en parallèle (comme la 1.0). Et notez bien la différence entre la capacité annoncée et l'espace de nom utilisé qui fait référence à une URN identifiant l'espace de nommage de NETCONF.



Le serveur fait alors de même et annonce ses capacités ou fonctionnalités. Ici, on a en plus l'identifiant de session. Il est important de voir que ces messages sont envoyés de façon asynchrone, le serveur n'attend pas sur le client pour envoyer le message Hello puisque les capacités de l'un et l'autre sont indépendantes.

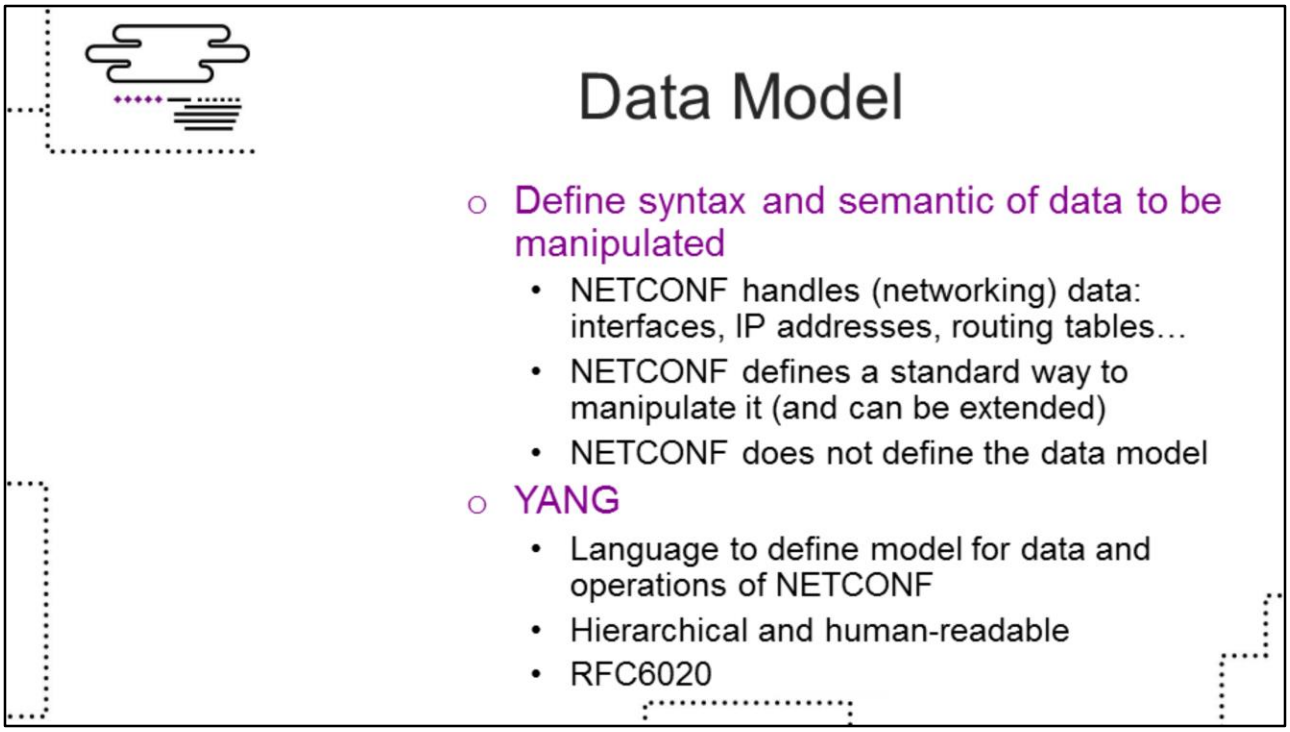


Une fois cela fait, le client peut donc envoyer des requêtes, puisqu'il a connaissance des capacités du serveur, et le serveur lui répond avec une réponse rpc-reply.
Nous allons bien entendu voir un peu plus en détail ce que contiennent ces messages.



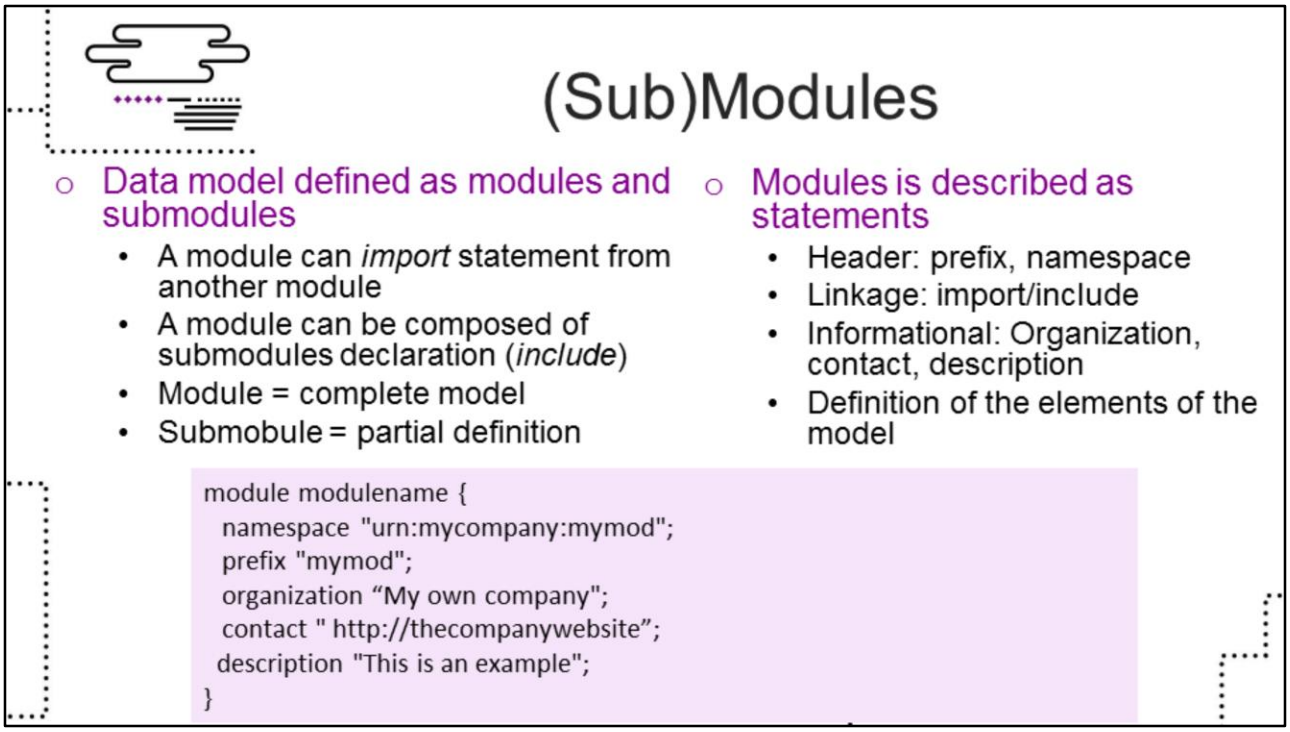
Pour résumer, dans les messages HELLO, et en fait dans tous les messages NETCONF, on lui spécifie l'espace de nommage à utiliser donc en principe celui de base mais qui peut bien sûr être étendu et complété. Cet espace de nommage définit les différents éléments de configuration que l'on va pouvoir ensuite manipuler et sert également à valider que la description XML fait bien référence à des éléments existants

Ici, vous voyez un exemple de capacité qui était mentionnée dans la diapositive précédente. Cette capacité "startup" signifie que l'entité configurée peut sauvegarder une configuration de démarrage. Celle-ci sera donc instanciée en running config au démarrage et évite donc de repartir d'une configuration vierge à chaque fois.



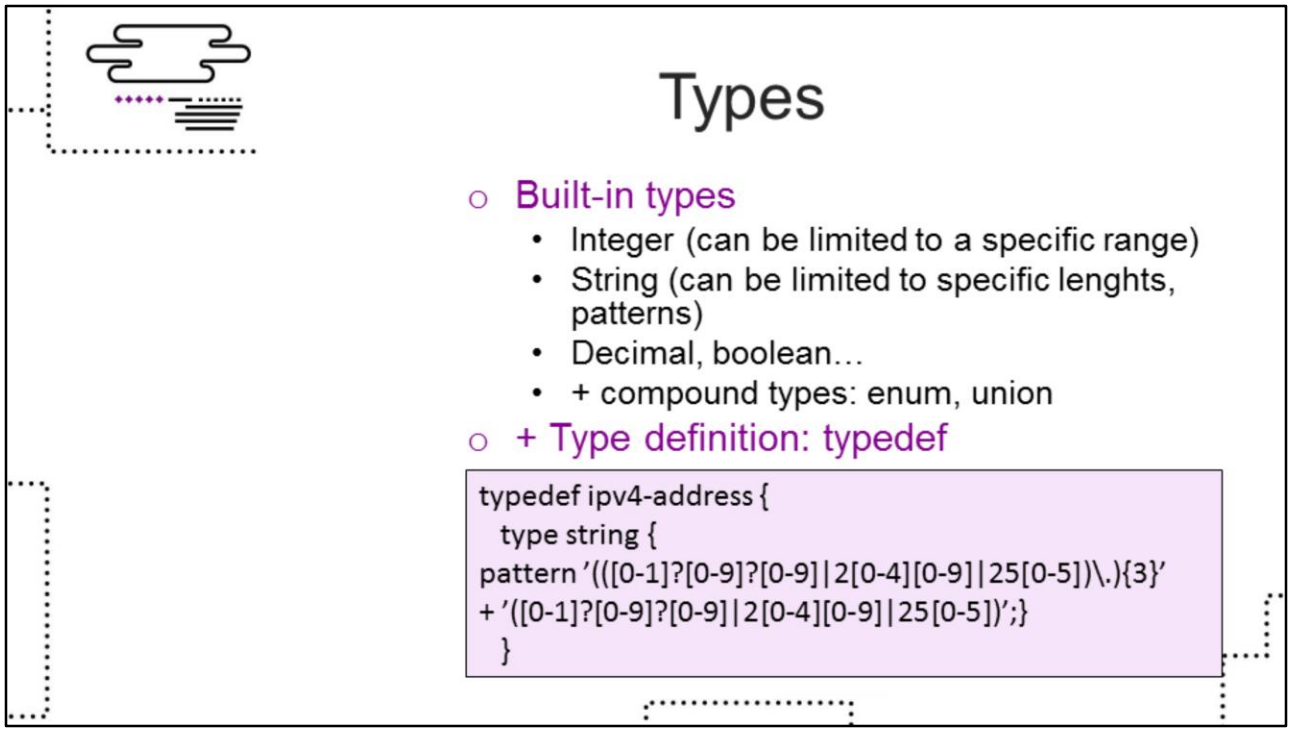
Avant d'aller plus loin dans les opérations réalisables, il est nécessaire de comprendre quel est le langage utilisé. Pour réaliser des opération,. NETCONF a donc besoin d'un langage pour manipuler les différents types de donnée,s principalement réseau comme les adresses IP, les ports ou les tables de routage. Comme annoncé dans les besoins, on se rend bien évidemment compte que ces éléments ou attributs sont communs à beaucoup de cas, équipements ou services. NETCONF permettra donc de les manipuler mais ne les définit pas. Le langage ou modèle de données est lui défini par YANG.

Je vous conseille vivement de consulter le RFC6020 pour une description détaillée de YANG. En quelques mots, le but de YANG est de définir, d'une manière simple, les différents éléments que NETCONF va manipuler, c'est-à-dire d'une manière facilement compréhensible par un humain et d'une façon hiérarchique, car effectivement nous avons tous l'habitude des classifications hiérarchiques.



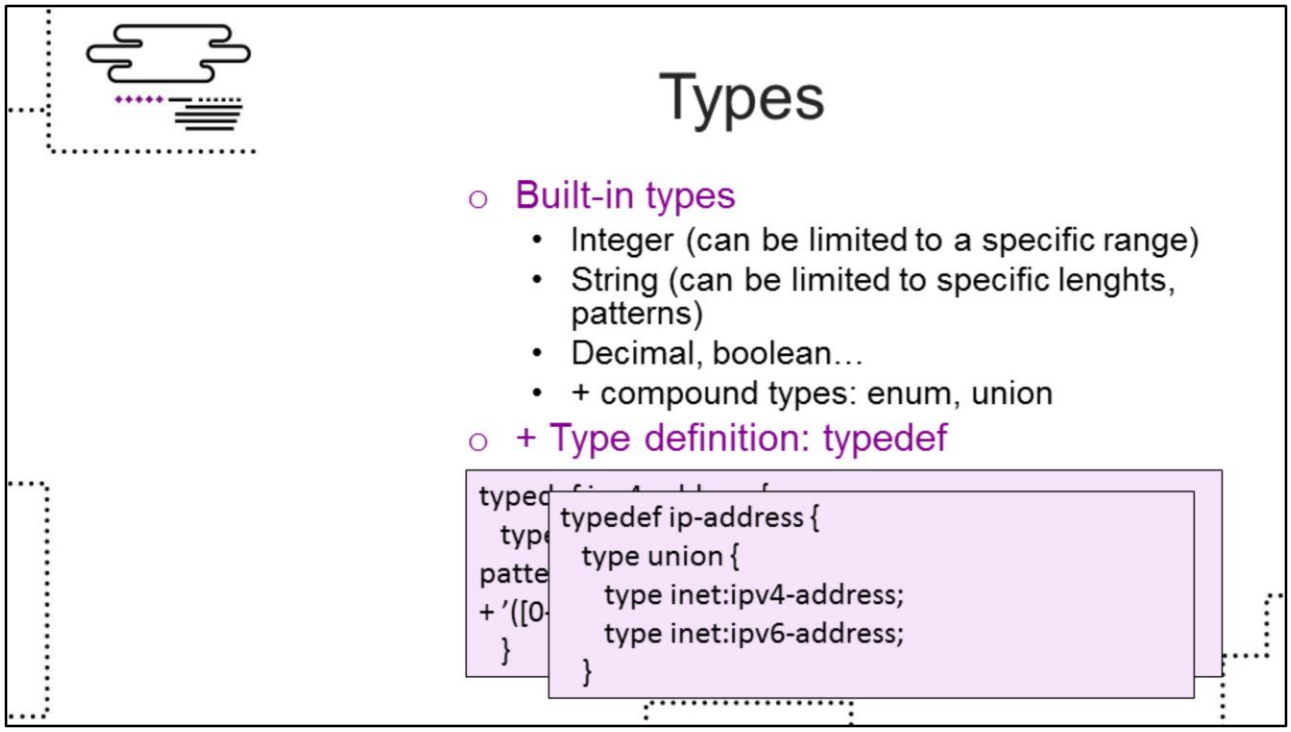
Dans YANG, tout est défini par des modules. Avec import, un module peut faire référence à un module externe défini dans une autre ressource et donc inclure les définitions de ce dernier. La déclaration include permet quant à elle de décomposer son module en plusieurs sous-modules

Ici, voici comment commence généralement un module. On fait d’abord référence à un espace de nommage et puis on trouve quelques informations générales sur le module, notamment celui qui l’a créé, un contact et surtout une description qui est un résumé texte de ce qui est défini dans le module. Cela correspond à l’entête d’un module et c’est aussi ici que l’on peut faire référence aux autres modules à inclure ou à importer. Mais jusque là, rien n‘est encore vraiment défini.

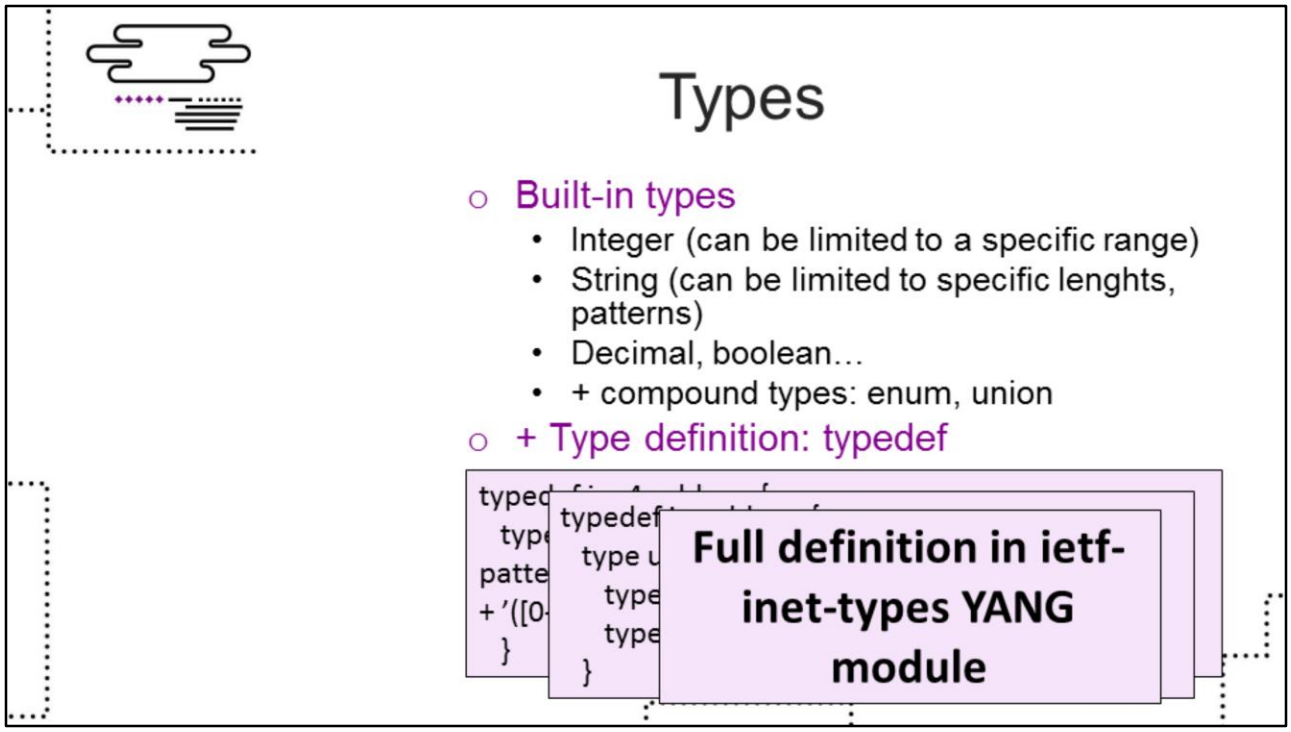


Pour la partie définition, on va naturellement s'appuyer sur des types prédéfinis simples comme les entiers, les chaînes de caractères ou les booléens. De la même façon qu'en programmation vous pouvez ensuite définir vos propres types plus complexes, types composés, à partir de ces derniers. Et c'est notamment l'utilisation de typedef qui permet alors de définir et nommer votre nouveau type…. Rien de bien compliqué ici mais juste un peu de syntaxe.

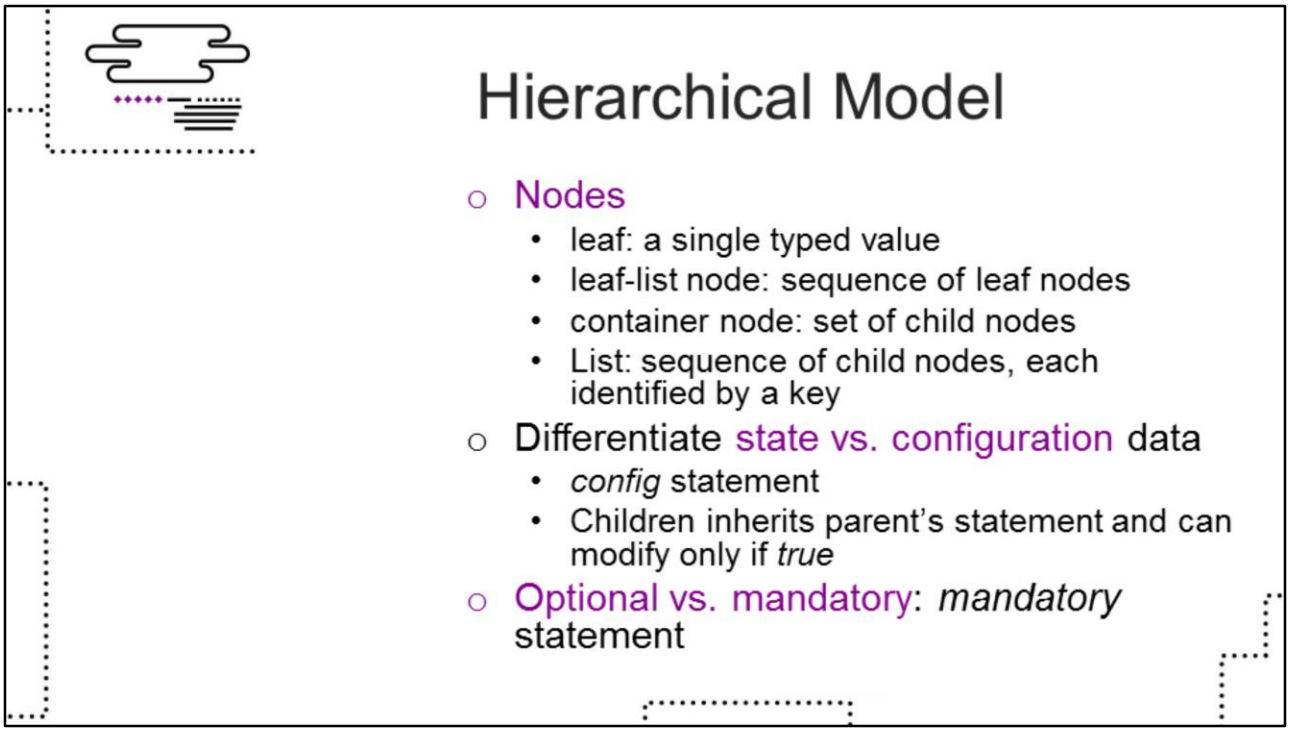
Par exemple, regardons comment définir une adresse IP. Une adresse IPv4 en notation facilement lisible est en notation pointée comme vous avez l'habitude d'en voir et c'est donc une chaîne de caractère avec un certain motif que l'on peut exprimer sous forme d'une expression régulière, car comme mentionnée dans cette diapositive, on peut effectivement bornée l'utilisation d'un type prédéfini à un sous-ensemble.



Mais finalement une adresse IP peut être en IPv6, on peut donc exactement faire la même chose dans ce cas là. Reste maintenant à spécifier qu'une adresse peut soit être en v4 soit en v6. Ici on utilise alors le mot clé union.



Vous pouvez consulter la définition complète de ce module qui s'appelle ietf-inet-types sur netconfcentral.org qui recense les différents modules prédéfinis et bien d'autres ressources sur le sujet. Aussi le RFC6021 décrit quelques types les plus utilisés.



Il y a quelques minutes, je vous ai parlé d'un modèle hiérarchique. En effet, imaginez un équipement configurable, l'ensemble des paramètres accessibles va être décrit sous une forme arborescente. On a donc des feuilles ou leaf en anglais qui vont représenter une et une seule variable.

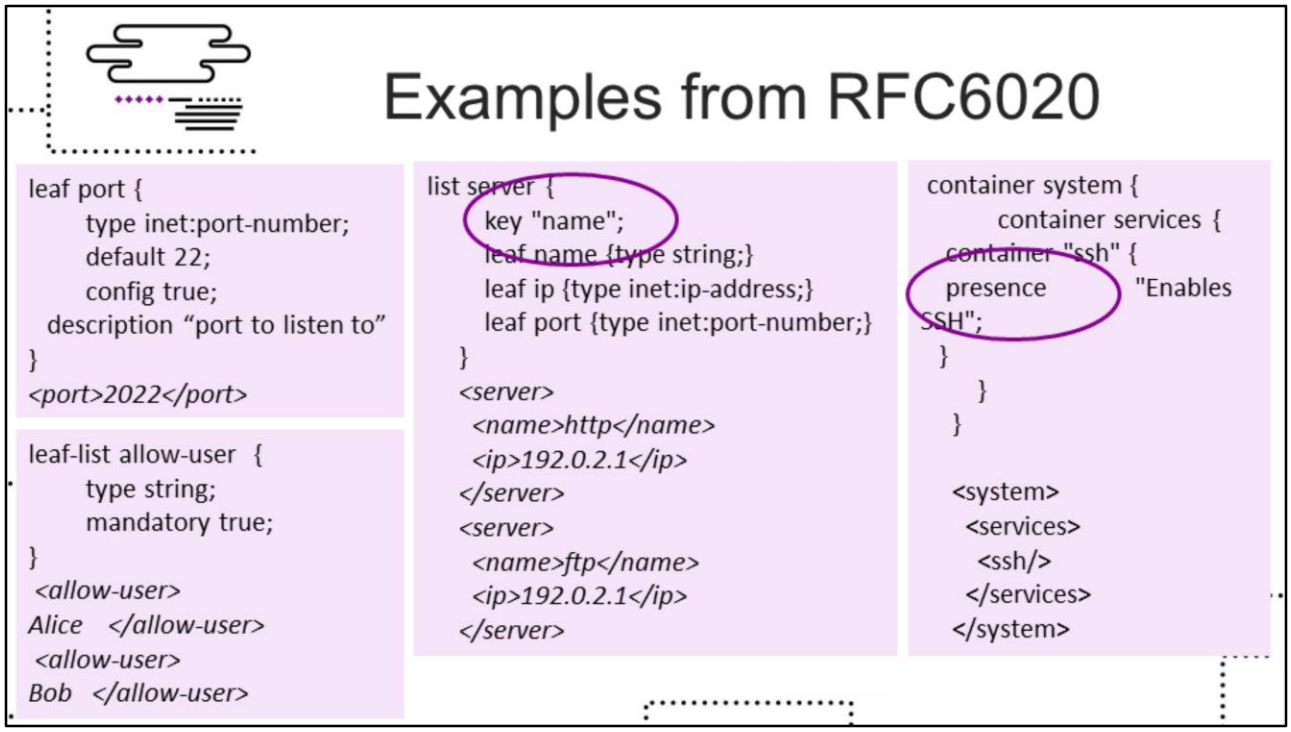
Il faut ensuite bien comprendre la notion de leaf-list qu'on pourrait traduire par une feuille liste et non une liste de feuilles. En effet il s'agit bien ici d'un autre type de feuille. La différence avec la première est qu'elle décrit une liste de valeurs, ou variables, mais toute du même type. On peut donc comparer cela à une liste d'objets typés en programmation. On a donc des nœuds feuilles unitaires ou des listes de valeurs du même type.

Jusque maintenant, nous ne pouvons donc définir que des feuilles. Pour les assembler et donc former une structure hiérarchique, il existe deux autres types de nœuds, les conteneurs et les listes à ne pas confondre avec les feuille liste.

La différence entre containeurs et listes est analogique de celles des types de feuilles. Un containeur est donc un ensemble d'autre nœuds où chacun d'entre eux ne sera donc au maximum présent qu'une seule fois. A l'inverse une liste représente une multitude d'éléments mais qui, cette fois, ne sont pas du même type mais ont la même structure, c'est plus ou moins une liste de conteneurs. Mais ne vous inquiétez pas, nous allons illustrer ces différences par des exemples.

Quelques mots clés importants, premièrement, config qui permet de spécifier si oui on non la définition concerne quelque chose de configurable (une donnée de configuration) comme une addresse IP ou plutôt une donnée d'état comme un compteur. Par défaut tout est configurable mais un nœud non configurable ne peut pas naturellement contenir de nœud configurable.

De la même façon, le mot clé mandatory permet de spécifier si, oui ou non, un nœud doit être présent dans l'arborescence ou s'il est seulement facultatif.



Premièrement, la définition d’un port, avec son type déjà défini dans le module ietf-inet-types. On spécifie ici que c’est une variable de configuration. Sa valeur par défaut est fixée à 22 qui correspond au service SSH en TCP. Un champ de description permet quant à lui de documenter la définition de l’élément. En dessous vous avez un exemple de configuration avec le langage XML.

Deuxième exemple, la fameuse feuille liste. Nous avons donc une liste d’utilisateurs de type chaîne de caractères. On a alors dans l’exemple des éléments d’une même liste et, comme vous pouvez l’observer, ce n’est pas une structure arborescente mais bien une liste à plat.

Troisième exemple, un nœud liste qui n’est donc plus une feuille mais un nœud interne de l’arborescence. Ici on définit une liste dont les éléments contiennent un nom, une adresse IP et un port. En fait, c’est plutôt que chaque élément peut contenir ces informations ou non puisque, par défaut, elles ne sont pas obligatoires. Ici le port n’apparaît pas.

Petite exception tout de même, cela ne s’applique pas pour la clé spécifiée ici par le mot clé « key ». Celle-ci identifie de façon unique chaque entrée dans la liste. Cela est en particulier utile pour récupérer une entrée de la liste en spécifiant sa clé. De façon générale une clé peut être composée de plusieurs champs de la liste et ces

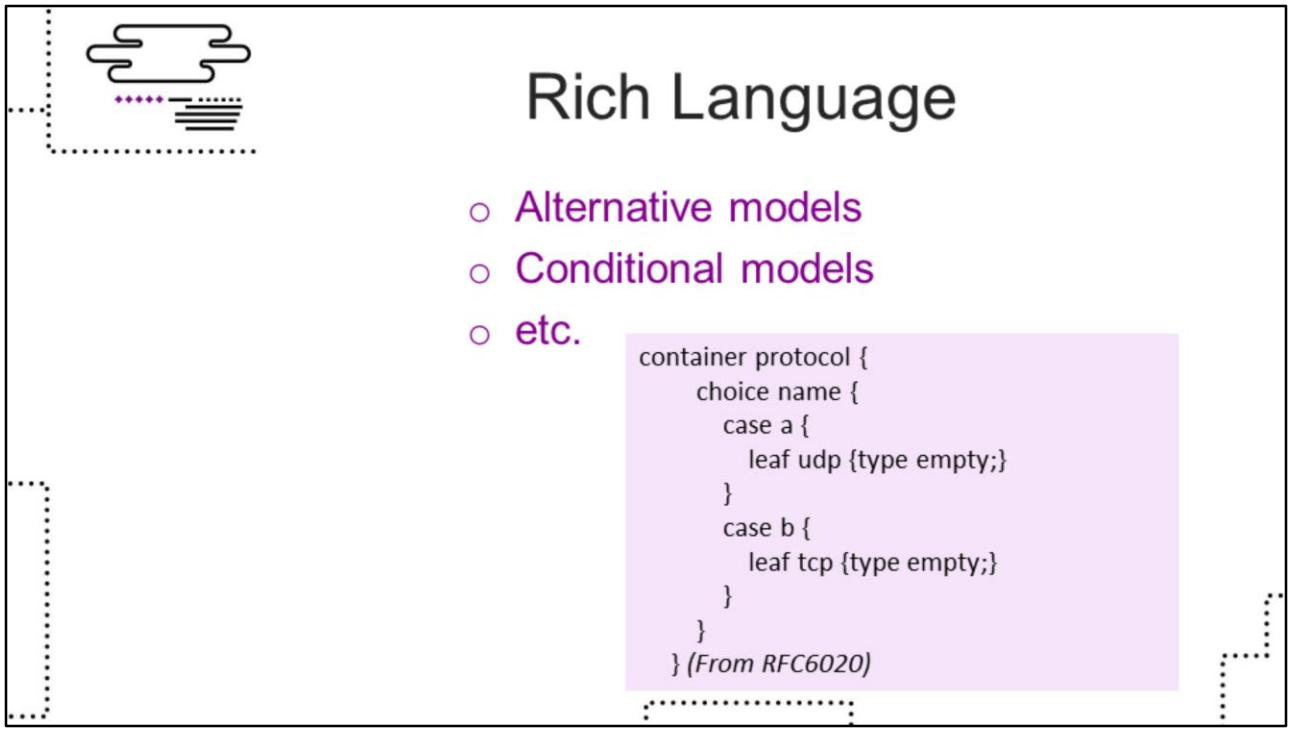
derniers doivent alors spécifiées dans le champ « key » séparés par un espace.

Dernier exemple, le conteneur ! Ici un exemple très basique ou l'on définit un système qui contient des services et ces derniers contiennent le service « ssh ». On pourrait définir ici une multitude de services et ensuite les avoir ou non dans la configuration.

Il y a encore une petite particularité. On peut distinguer deux types de conteneurs, ceux qui ne servent qu'à organiser le contenu des informations pour rendre l'accès et la modifications des données plus simples et ceux qui auront une répercussion sur la configuration, c'est-à-dire ceux qui ont un sens et un impact sur la configuration.

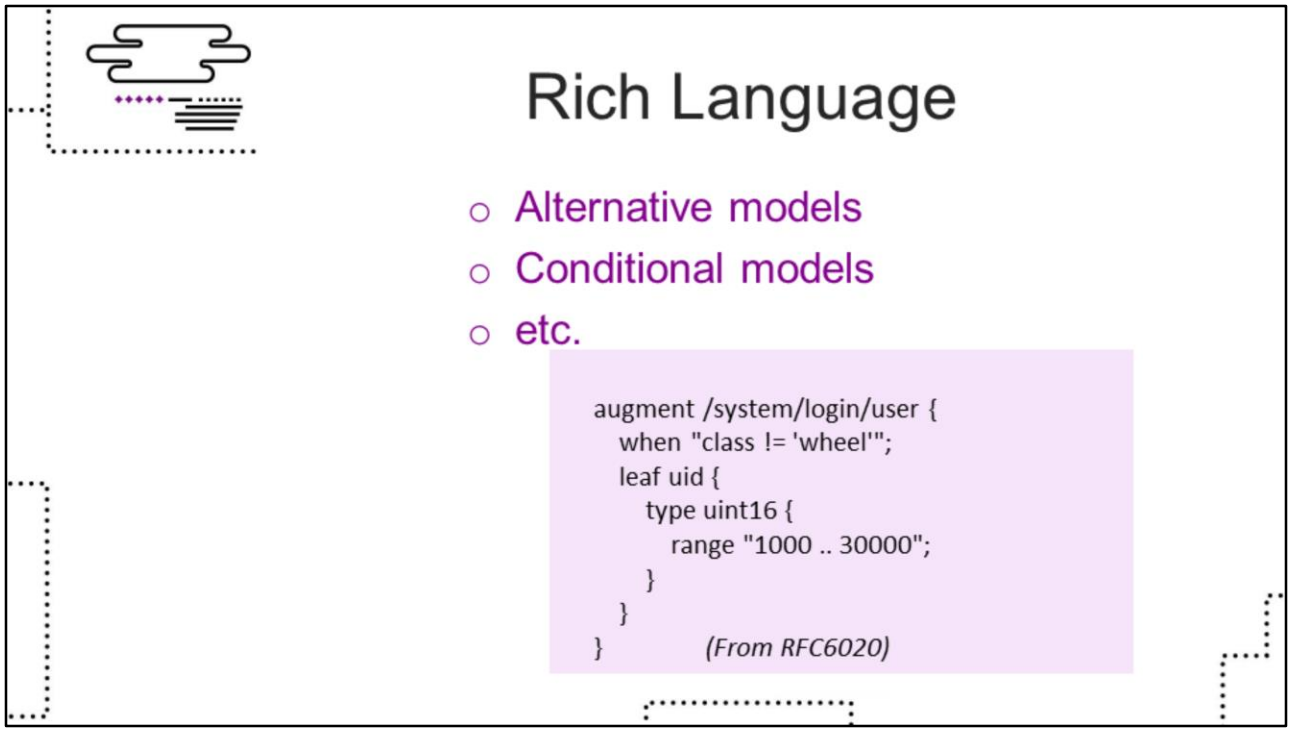
C'est le cas ici du conteneur « SSH » dont la présence, comme vous pouvez le voir ici, signifie que le service SSH doit être activé. Comme je le disais un instant, on pourrait donc définir plusieurs services comme ftp, http, dns et ainsi en les ajoutant ou non dans la configuration (c'est-à-dire en les ajoutant dans l'XML), les activer.

Il ne faut bien entendu pas confondre presence et mandatory.



Ce premier cours sur NETCONF se termine donc sur le langage YANG qui permet de modéliser les différentes données d'état ou de configuration qui seront ensuite utilisées par le protocole NETCONF, ce que nous verrons dans le prochain cours.

Je tiens néanmoins à attirer votre attention que la richesse du langage YANG est bien plus grande que ces quelques diapositives et vous invite une fois de plus à consulter le RFC6020.
A titre de complément, sachez qu'il existe aussi un type vide, empty. Une variable de ce type ne contient que de l'information par sa presence ou non. Il est aussi possible de définir une structure de données alternative avec le mot clé case. Ici, le conteneur protocole est défini par un nom qui peut être soit le cas a ou le cas b, respectivement pour spécifier tcp ou udp.



Encore un autre exemple avancé, où dans un module on peut importer un autre module, et même modifier le modèle de données associé, dans le cas présent en l'augmentant et en utilisant une définition conditionnelle. Ainsi, si l'utilisateur n'est pas de la classe "wheel", alors l'utilisateur aura également à associer un identifiant (user id).