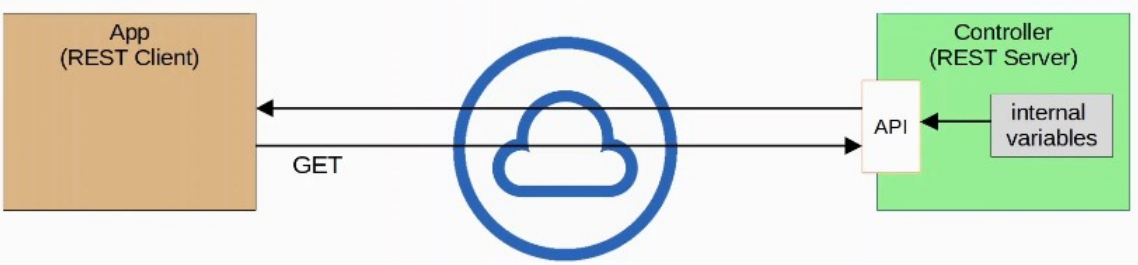
Cours 60 : JSON, XML, & YAML

Dans ce cours nous verrons ce qu'est JSON, XML et YAML. Ce sont des langages connus pour la sérialisation des données, ou le formatage de sérialisation des données. Ceci nous permet de formater ou structurer les données de manière standard pour que plusieurs applications puissent communiquer entre elles. Nous verrons tout d'abord ce qu'est la sérialisation des données, puis JSON (JavaScript Object Notation) comment l'interpréter et l'utiliser, nous verrons aussi XML (Extensible Markup Language) et YAML (YAML Ain't Markup Language).

La sérialisation des données est le processus de conversion de données dans un format ou une structure standard qui peut être stocké (dans un fichier) ou transmis (par le réseau) et reconstruit plus tard (par exemple par une différente application). Cela est utile car cela permet aux données d'être communiqués entre plusieurs applications d'une manière à ce que les deux applications comprennent. Le langage de sérialisation des données permet de représenter des variables avec du texte.
Voici un exemple de variable :

```
{"interface_name": "GigabitEthernet1/1",
 "status": "up",
 "ip_address": "192.168.1.1",
 "netmask": "255.255.255.0"}
```

Il y a ici 4 variables, des variables contiennent et stockent des valeurs. Par exemple « Interface_Name » est un conteneur qui contient la valeur « gigabitethernet1/1 ».
Voyons comment l'échange de données devrait marcher sans utiliser la sérialisation standard des données.



Ci dessous l'application envoie une requête GET vers le controller, le controller réceptionne la demande par l'API et répond à sa requête.
Pour que les deux puissent communiquer (l'appli et le contrôleur) et se comprendre, le langage avec la sérialisation des données est utilisé. Le client envoie sa requête et est convertie par l'API en un format standard JSON puis est envoyé sur le réseau du contrôleur.

JSON (JavaScript Object Notation) est un standard open en format de fichier et un format de données interchangeable qui utilise un texte lisible par l'humain qui stocke et transmet les données des objets. JSON a été standardisé dans le RFC 8259. JSON a été dérivé de JavaScript mais est un langage indépendant et plusieurs langages de programmation modernes sont capables de générer et lire des données JSON. REST API utilise souvent JSON. Les espaces ne sont pas significatifs dans JSON ils ne changent pas le sens des données. Il y a quatre type de données primitif : string, nombre, booléen, nul. Il y a aussi deux type de données structurés : objets et tableau.

- Un string est une valeur dans un texte, il est entouré par des double guillemets « ».
Par exemple « Hello » est un string.
- Un nombre est une valeur numérique et n'est par entouré par des doubles guillemets.
Par exemple 5, 10 sont des nombres.
- Un booléen est un type de données qui a seulement deux valeurs possibles, et non entouré par des guillemets : true et false (Vrai ou Faux) et écris en minuscule.
- Une valeur nul est l'absence intentionnel de n'importe quelle objet de valeur et n'est pas entouré de guillemets.

Le type structuré des données JSON est ainsi :
Un objet est une liste non ordonnée par une pair de valeur clés (Les variables).
Les objets sont entourés par des accolades ({}). La clé est un string. La valeur est n'importe quelle type de données valide en JSON (string, nombre, booléen, nul, objet, tableau).
La clé et la valeur sont séparés par des deux point « : »
S'il y a plusieurs pairs de valeurs clés, chaque pair est séparé par une virgule.
Voici un exemple en bleue la variable et en rouge la valeur associé :

```
{
 "interface": "GigabitEthernet1/1",
 "is_up": true,
 "ipaddress": "192.168.1.1",
 "netmask": "255.255.255.0",
 "speed": 1000
}
```

Il est aussi possible d'écrire de la manière suivante car les espaces ne comptent pas :

```
{"interface":"GigabitEthernet1/1","is_
up":true,"ipaddress":"192.168.1.1","ne
tmask":"255.255.255.0","speed":1000}
```

Comme on peut le voir dans l'exemple suivant, les objets sont des types de données valides d'un pair de valeur clé :

```
{
    "device": {
       "name": "R1",
       "vendor": "Cisco",
       "model": "1101"
    },

    "interface_config": {
       "interface_name": "GigabitEthernet1/1",
       "is_up": true,
       "ipaddress": "192.168.1.1",
       "netmask": "255.255.255.0",
       "speed": 1000
    }
}
```

Ici la clé est « device » et « interface_config ». La valeur ou objet est contenu dans les accolades.

Un tableau ou « array » est une série de valeurs séparés par une virgule.
Il n'y a pas de valeur clé, c'est seulement une série de valeurs. Il n'est pas nécessaire que les valeurs soient du même type de données.
En voici un exemple :

```
{
    "interfaces": [
        "GigabitEthernet1/1",
        "GigabitEthernet1/2",
        "GigabitEthernet1/3"
    ],

    "random_values": [
        "Hi",
        5
    ]
}
```

Voici un résultat de la commande : « show interface brief » sur un Routeur :

```
R1#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     192.168.1.1     YES manual up                    up
GigabitEthernet0/1     unassigned      YES unset  administratively down down
```

Voici le résultat de la même commande au format JSON :

```
{
    "ip_interfaces": [
            {
            "Interface":"GigabitEthernet0/0",
            "IP-Address": "192.168.1.1",
            "OK?": "YES",
            "Method": "manual",
            "Status": "up",
            "Protocol": "up"
            },
            {
            "Interface":"GigabitEthernet0/1",
            "IP-Address": "unassigned",
            "OK?": "YES",
            "Method": "unset",
            "Status": "administratively down",
            "Protocol": "down"
            }
    ]
}
```

Certaines fois un objet est appelé un dictionnaire.

XML est l'acronyme de (Extensible Markup Language) il a été développé comme langage de Markup, mais est maintenant utilisé comme langage général de sérialisation de données.
Les langages Markup (Par exemple HTML) sont utilisé pour formater le texte (Le font, la taille, la couleur, l'entête, etc...) XML est en général moins facilement lisible pour les humain par rapport à JSON. Les espaces ne sont pas significatifs et ne changent pas le sens du texte. XML est souvent utilisé par des REST API. Le format général qui est utilisé est : <key>valeur</key> par exemple :

```
R1#show ip interface brief | format
<?xml version="1.0" encoding="UTF-8"?>
  <ShowIpInterfaceBrief xmlns="ODM://built-in//show_ip_interface_brief">
    <SpecVersion>built-in</SpecVersion>
    <IPInterfaces>
      <entry>
        <Interface>GigabitEthernet0/0</Interface>
        <IP-Address>192.168.1.1</IP-Address>
        <OK>YES</OK>
        <Method>manual</Method>
        <Status>up</Status>
        <Protocol>up</Protocol>
      </entry>
      <entry>
        <Interface>GigabitEthernet0/1</Interface>
        <OK>YES</OK>
        <Method>unset</Method>
        <Status>administratively down</Status>
        <Protocol>down</Protocol>
      </entry>
    </IPInterfaces>
  </ShowIpInterfaceBrief>
```

on peut comparer les deux format de langage directement depuis la ligne de commandes :

```
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         192.168.1.1     YES manual up                    up
GigabitEthernet0/1         unassigned      YES unset  administratively down down


R1#show ip interface brief | format
<?xml version="1.0" encoding="UTF-8"?>
  <ShowIpInterfaceBrief xmlns="ODM://built-in//show_ip_interface_brief">
    <SpecVersion>built-in</SpecVersion>
    <IPInterfaces>
      <entry>
        <Interface>GigabitEthernet0/0</Interface>
        <IP-Address>192.168.1.1</IP-Address>
        <OK>YES</OK>
        <Method>manual</Method>
        <Status>up</Status>
        <Protocol>up</Protocol>
      </entry>
      <entry>
        <Interface>GigabitEthernet0/1</Interface>
        <OK>YES</OK>
        <Method>unset</Method>
        <Status>administratively down</Status>
        <Protocol>down</Protocol>
      </entry>
    </IPInterfaces>
  </ShowIpInterfaceBrief>
```

YAML signifie à l'origine « Yet Another Markup Language », mais n'est pas différents du langage de sérialisation plutôt qu'un langage de Markup, il a été renommé : YAML Ain't Markup Language.
YAML est utilisé par le réseau pour l'automatisation du réseau avec l'outil Ansible.
YAML est facilement lisible par l'humain. Les espace sont signifiant, c'est à dire qu'ils changent l'interprétation du langage, l'indentation est donc très importante.
Les fichier YAML commencent par des ---
Un seule – est utilisé pour indiquer une liste.
Les valeurs et clefs sont représenté au format key:valeur

```
---
ip_interfaces:
- Interface: GigabitEthernet0/0
  IP-Address: 192.168.1.1
  OK?: 'YES'
  Method: manual
  Status: up
  Protocol: up
- Interface: GigabitEthernet0/1
  IP-Address: unassigned
  OK?: 'YES'
  Method: unset
  Status: administratively down
  Protocol: down
```

Voici une comparaison de texte entre JSON et YAML :

JSON

```
{
    "ip_interfaces": [
            {
            "Interface":"GigabitEthernet0/0",
            "IP-Address": "192.168.1.1",
            "OK?": "YES",
            "Method": "manual",
            "Status": "up",
            "Protocol": "up"
            },
            {
            "Interface":"GigabitEthernet0/1",
            "IP-Address": "unassigned",
            "OK?": "YES",
            "Method": "unset",
            "Status": "administratively down",
            "Protocol": "down"
            }
    ]
}
```

YAML

```
---
ip_interfaces:
- Interface: GigabitEthernet0/0
  IP-Address: 192.168.1.1
  OK?: 'YES'
  Method: manual
  Status: up
  Protocol: up
- Interface: GigabitEthernet0/1
  IP-Address: unassigned
  OK?: 'YES'
  Method: unset
  Status: administratively down
  Protocol: down
```