

CCNA Day 60

JSON, XML, & YAML

6.0 Automation and Programmability

10%



- 6.1 Explain how automation impacts network management
- 6.2 Compare traditional networks with controller-based networking
- 6.3 Describe controller-based and software defined architectures (overlay, underlay, and fabric)
 - 6.3.a Separation of control plane and data plane
 - 6.3.b North-bound and south-bound APIs
- 6.4 Compare traditional campus device management with Cisco DNA Center enabled device management
- 6.5 Describe characteristics of REST-based APIs (CRUD, HTTP verbs, and data encoding)
- 6.6 Recognize the capabilities of configuration management mechanisms Puppet, Chef, and Ansible
- 6.7 Interpret JSON encoded data



Things we'll cover

- Data Serialization
- JSON (JavaScript Object Notation)
- XML (Extensible Markup Language)
- YAML (YAML Ain't Markup Language)

Data Serialization

- Data serialization is the process of converting data into a standardized format/structure that can be stored (in a file) or transmitted (over a network) and reconstructed later (ie. by a different application).
→ This allows the data to be communicated between applications in a way both applications understand.

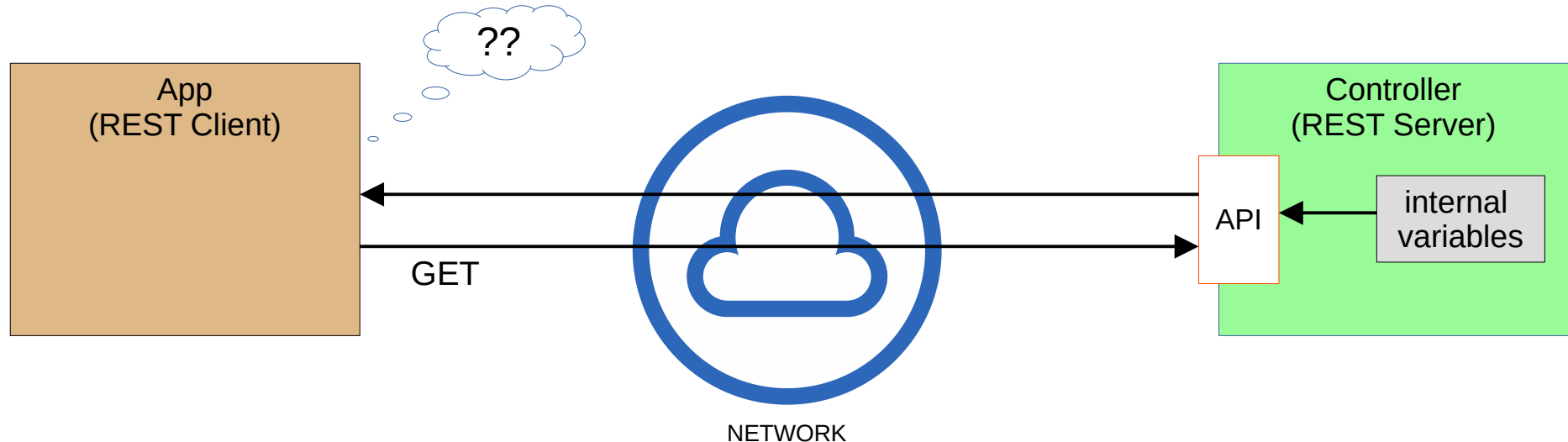
- Data serialization languages allow us to represent *variables* with text.

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

Variables are containers that store values.

"interface_name" = container

"GigabitEthernet1/1" = value



Data Serialization

- Data serialization is the process of converting data into a standardized format/structure that can be stored (in a file) or transmitted (over a network) and reconstructed later (ie. by a different application).
→ This allows the data to be communicated between applications in a way both applications understand.

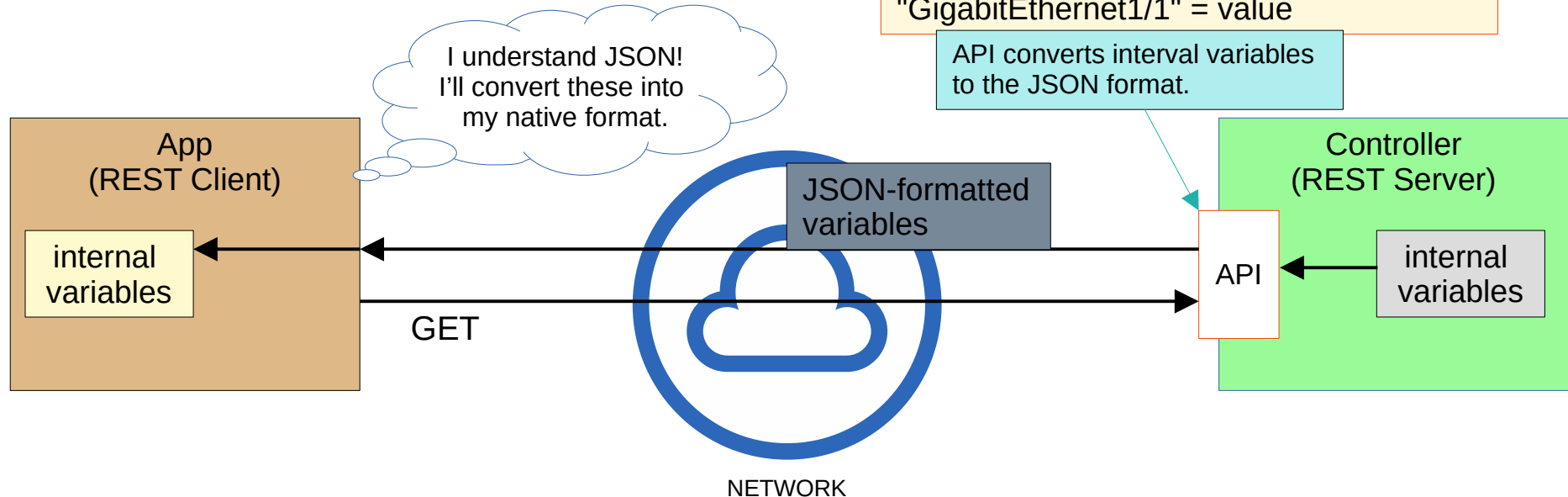
- Data serialization languages allow us to represent *variables* with text.

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

Variables are containers that store values.

"interface_name" = container
"GigabitEthernet1/1" = value

API converts internal variables to the JSON format.



JSON

- **JSON** (JavaScript Object Notation) is an open standard **file format** and **data interchange format** that uses human-readable text to store and transmit data objects.
- It is standardized in RFC 8259 (<https://datatracker.ietf.org/doc/html/rfc8259>).
- It was derived from JavaScript, but it is language-independent and many modern programming languages are able to generate and read JSON data.
→ REST APIs often use JSON.
- *Whitespace* is insignificant.
- JSON can represent four 'primitive' data types:
 - string
 - number
 - boolean
 - null
- JSON also has two 'structured' data types:
 - object
 - array



Take the time to read it!

JSON primitive data types:

- A **string** is a text value. It is surrounded by double quotes " " .
 - "Hello."
 - "five"
 - "5"
 - "true"
 - "null"
- A **number** is a numeric value. It is not surrounded by quotes.
 - 5
 - 100
- A **boolean** is a data type that has only two possible values, not surrounded by quotes:
 - true
 - false
- A **null** value represents the intentional absence of any object value. It is not surrounded by quotes.
 - null

JSON structured data types:

- An **object** is an unordered list of *key-value pairs* (variables).
 - Objects are surrounded by curly brackets {} .
 - The *key* is a string.
 - The *value* is any valid JSON data type (string, number, boolean, null, object, array).
 - The *key* and *value* are separated by a colon : .
 - If there are multiple key-value pairs, each pair is separated by a comma.

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

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

These two are the same! In JSON, whitespace (spaces etc.) is insignificant.

JSON structured data types:

- An **object** is an unordered list of *key-value pairs* (variables).
 - Objects are surrounded by curly brackets {} .
 - The *key* is a string.
 - The *value* is any valid JSON data type (string, number, boolean, null, object, array).
 - The *key* and *value* are separated by a colon : .
 - If there are multiple key-value pairs, each pair is separated by a comma.

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


As shown in the following example, objects are a valid data type for the value of a key-value pair:

```
{
  "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
  }
}
```

key value (object)

key value (object)

Objects within objects are called 'nested objects'.

JSON structured data types:

- An **object** is an unordered list of *key-value pairs* (variables).
- An **array** is a series of *values* separated by commas.
 - not *key-value pairs*.
 - the values don't have to be the same data type.

```
{  
  "interfaces": [  
    "GigabitEthernet1/1",  
    "GigabitEthernet1/2",  
    "GigabitEthernet1/3"  
  ],  
  "random_values": [  
    "Hi",  
    5  
  ]  
}
```

```
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

```
{
  "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"
    }
  ]
}
```

JSON primitive data types:

- A **string** is a text value. It is surrounded by double quotes " " .
- A **number** is a numeric value. It is not surrounded by quotes.
- A **boolean** is a data type that has only two possible values, not surrounded by quotes.
- A **null** value represents the intentional absence of any object value. It is not surrounded by quotes.

JSON structured data types:

- An **object** is an unordered list of *key-value pairs* (variables).
→ Sometimes called a **dictionary**.
- An **array** is a series of *values* separated by commas.

- **XML** (Extensible Markup Language) was developed as a markup language, but is now used as a general data serialization language.
→ markup languages (ie. HTML) are used to format text (font, size, color, headings, etc.)
- XML is generally less human-readable than JSON.
- Whitespace is insignificant.
- Often used by REST APIs.
- `<key>value</key>`

```
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>
```

```
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** originally meant *Yet Another Markup Language*, but to distinguish its purpose as a data-serialization language rather than a markup language, it was repurposed to *YAML Aint Markup Language*.
- YAML is used by the network automation tool Ansible (we'll cover that later!).
- YAML is very human-readable.
- Whitespace **is significant** (unlike JSON and XML).
 - Indentation is very important.
- YAML files start with `---`.
- `-` is used to indicate a list.
- Keys and values are represented as `key:value`.

```
---
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
```

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
```


Things we covered

- Data Serialization
- JSON (JavaScript Object Notation)
- XML (Extensible Markup Language)
- YAML (YAML Ain't Markup Language)

In which of the following data serialization languages is whitespace significant?

- a) JSON
- b) YAML
- c) XML
- d) All of the above

Which of the following data serialization languages formats key-value pairs like this?

`<key>value</key>`

- a) JSON
- b) YAML
- c) XML
- d) All of the above

Which of the following is NOT a valid JSON data type?

- a) object
- b) string
- c) number
- d) boolean
- e) key
- f) array
- g) null

Examine the JSON-formatted data. Which of the following statements is true?

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

- a) It is valid JSON data.
- b) A curly bracket is missing.
- c) There are too many commas.
- d) There are too many colons.

Quiz 5

Examine the JSON-formatted data. Which of the following statements is true?

- a) The value of "ip_interfaces" is an object.
- b) The value of "ip_interfaces" is an array.
- c) The value of "ip_interfaces" is multiple objects.
- d) The value of "ip_interfaces" is a string.

```
{
  "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"
    }
  ]
}
```

Which of the following is an example of valid JSON-formatted data?

a)

```
{
  "interfaces", [
    "GigabitEthernet1/1",
    "GigabitEthernet1/2",
    "GigabitEthernet1/3"
  ],
  "random_values": [
    "Hi",
    5
  ]
}
```

b)

```
{
  "interfaces": [
    "GigabitEthernet1/1",
    "GigabitEthernet1/2",
    "GigabitEthernet1/3"
  ],
  "random_values": [
    "Hi",
    5:
  ]
}
```

c)

```
{
  "interfaces": [
    "GigabitEthernet1/1",
    "GigabitEthernet1/2",
    "GigabitEthernet1/3"
  ],
  "random_values": [
    "Hi",
    5
  ]
}
```

d)

```
{
  "interfaces": [
    "GigabitEthernet1/1",
    "GigabitEthernet1/2",
    "GigabitEthernet1/3"
  ],
  "random_values": [
    "Hi",
    5
  ]
}
```

Quiz 7

Examine the JSON-formatted data. Which of the following statements is true?

- a) The value of “is_up” is a boolean.
- b) It is not valid JSON data.
- c) The value of “ipaddress” is a number.
- d) The value of “speed” is a string.

```
{  
  "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  
  }  
}
```


Examine the JSON-formatted data. Which of the following statements is true?

- a) The whitespace formatting is invalid.
- b) A curly bracket is missing.
- c) A comma is missing.
- d) It is valid JSON data.

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

Examine the JSON-formatted data. Which of the following statements is true?

- a) It is valid JSON data.
- b) A curly bracket is missing.
- c) A square bracket is missing.
- d) The value of "version" is a string.

```
{
  "network": {
    "version": 2,
    "renderer": "networkd",
    "ethernets": {
      "enp3s0": {
        "addresses": [
          "10.10.10.2/24"
        ],
        "nameservers": {
          "search": [
            "mydomain",
            "otherdomain"
          ],
          "addresses": [
            "10.10.10.1",
            "1.1.1.1"
          ]
        }
      },
      "routes": [
        {
          "to": "default",
          "via": "10.10.10.1"
        }
      ]
    }
  }
}
```

```
{
  "network": {
    "version": 2,
    "renderer": "networkd",
    "ethernets": {
      "enp3s0": {
        "addresses": [
          "10.10.10.2/24"
        ],
        "nameservers": {
          "search": [
            "mydomain",
            "otherdomain"
          ],
          "addresses": [
            "10.10.10.1",
            "1.1.1.1"
          ]
        }
      },
      "routes": [
        {
          "to": "default",
          "via": "10.10.10.1"
        }
      ]
    }
  }
}
```

Examine the JSON-formatted data. Which of the following statements is true?

- a) It is valid JSON data.
- b) A curly bracket is missing.
- c) A square bracket is missing.
- d) The value of “dhcp4” is a boolean.

```
{  "network": {
    "version": 2,
    "renderer": "networkd",
    "wifis": {
      "wlp2s0b1": {
        "dhcp4": "no",
        "dhcp6": "no",
        "addresses": [
          "192.168.0.21/24"
        ],
        "nameservers": {
          "addresses": [
            "192.168.0.1",
            "8.8.8.8"
          ]
        }
      },
      "access-points": {
        "network_ssid_name": {
          "password": "*****"
        }
      }
    },
    "routes": [
      {
        "to": "default",
        "via": "192.168.0.1"
      }
    ]
  }
}
```