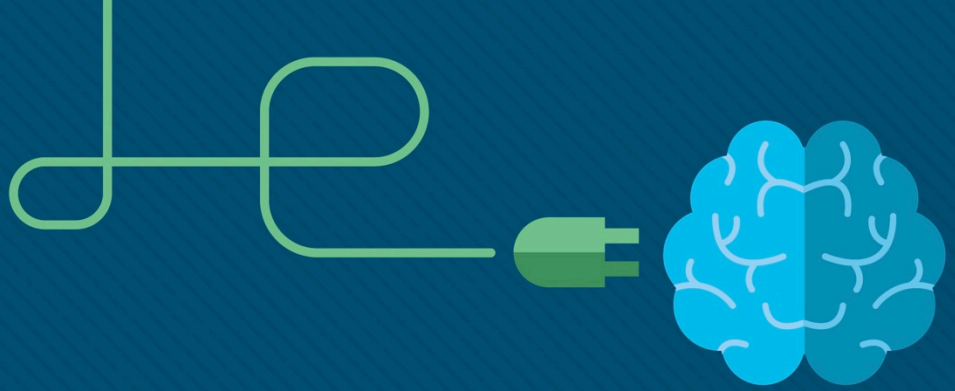


# Module 14: Network Automation

Enterprise Networking, Security, and Automation v7.0 (ENSA)

# Module Objectives

**Module Title:** Network Virtualization

**Module Objective**: Explain the purpose and characteristics of network virtualization.

| Topic Title | Topic Objective |
|---|---|
| **Cloud Computing** | Explain the importance of cloud computing. |
| **Virtualization** | Explain the importance of virtualization. |
| **Virtual Network Infrastructure** | Describe the virtualization of network devices and services. |
| **Software-Defined Networking** | Describe software-defined networking. |
| **Controllers** | Describe controllers used in network programming. |

# 14.1 Automation Overview

# The Increase in Automation

These are some of the benefits of automation:

- Machines can work 24 hours a day without breaks, which results in greater output.
- Machines provide a more uniform product.
- Automation allows the collection of vast amounts of data that can be quickly analyzed to provide information which can help guide an event or process.
- Robots are used in dangerous conditions such as mining, firefighting, and cleaning up industrial accidents. This reduces the risk to humans.
- Under certain circumstances, smart devices can alter their behavior to reduce energy usage, make a medical diagnosis, and improve automobile driving safety.

# Thinking Devices

- Many devices now incorporate smart technology to help to govern their behavior. This can be as simple as a smart appliance lowering its power consumption during periods of peak demand or as complex as a self-driving car.
- Whenever a device takes a course of action based on an outside piece of information, then that device is referred to as a smart device. Many devices that we interact with now have the word smart in their names. This indicates that the device has the ability to alter its behavior depending on its environment.
- In order for devices to “think”, they need to be programmed using network automation tools.

# 14.2 Data Formats

# The Data Formats Concept

- Data formats are simply a way to store and exchange data in a structured format. One such format is called Hypertext Markup Language (HTML). HTML is a standard markup language for describing the structure of web pages.
- These are some common data formats that are used in many applications including network automation and programmability:
  - JavaScript Object Notation (JSON)
  - eXtensible Markup Language (XML)
  - YAML Ain’t Markup Language (YAML)
- The data format that is selected will depend on the format that is used by the application, tool, or script that you are using. Many systems will be able to support more than one data format, which allows the user to choose their preferred one.

# Data Format Rules

Data formats have rules and structure similar to what we have with programming and written languages. Each data format will have specific characteristics:

- Syntax, which includes the types of brackets used, such as [ ], ( ), { }, the use of white space, or indentation, quotes, commas, and more.
- How objects are represented, such as characters, strings, lists, and arrays.
- How key/value pairs are represented. The key is usually on the left side and it identifies or describes the data. The value on the right is the data itself and can be a character, string, number, list or another type of data.

```
{"message": "success", "timestamp": 1560789216, "iss_position": {"latitude": "25.9990",
"longitude": "-132.6992"}}
```

# Compare Data Formats

```
{
    "message": "success",
    "timestamp": 1560789260,
    "iss_position": {
        "latitude": "25.9990",
        "longitude": "-132.6992"
    }
}
```

JSON Format

```
message: success
timestamp: 1560789260
iss_position:
    latitude: '25.9990'
    longitude: '-132.6992'
```

YAML Format

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
<message>success</message>
<timestamp>1560789260</timestamp>
<iss_position>
   <latitude>25.9990</latitude>
   <longitude>-132.6992</longitude>
</iss_position>
</root>
```

XML Format

# JSON Data Format

- JSON is a human readable data format used by applications for storing, transferring and reading data. JSON is a very popular format used by web services and APIs to provide public data. This is because it is easy to parse and can be used with most modern programming languages, including Python.

# JSON Data Format (Cont.)

```
GigabitEthernet0/0/0 is up, line protocol is up (connected)
   Description: Wide Area Network
   Internet address is 172.16.0.2/24
```

Compare the IOS output above to the output in JSON format. Notice that each object (each key/value pair) is a different piece of data about the interface including its name, a description, and whether the interface is enabled.

```
{
"ietf-interfaces:interface": {
    "name": "GigabitEthernet0/0/0",
    "description": "Wide Area Network”,
    "enabled": true,
    "ietf-ip:ipv4": {
        "address": [
            {
            "ip": "172.16.0.2",
            "netmask": "255.255.255.0"
            }
        ]
    }
  }
}
```

# JSON Syntax Rules

These are some of the characteristics of JSON:

- It uses a hierarchical structure and contains nested values.
- It uses braces { } to hold objects and square brackets [ ] hold arrays.
- Its data is written as key/value pairs.

With JSON, the data known as an object is one or more key/value pairs enclosed in braces { }. The syntax for a JSON object includes:

- Keys must be strings within double quotation marks " ".
- Values must be a valid JSON data type (string, number, array, Boolean, null, or another object).
- Keys and values are separated by a colon.
- Multiple key/value pairs within an object are separated by commas.
- White space is not significant.

# JSON Syntax Rules (Cont.)

At times a key may contain more than one value. This is known as an array. An array in JSON is an ordered list of values. Characteristics of arrays in JSON include:

- The key followed by a colon and a list of values enclosed in square brackets [ ].
- The array is an ordered list of values.
- The array can contain multiple value types including a string, number, Boolean, object or another array inside the array.
- Each value in the array is separated by a comma.

# JSON Syntax Rules (Cont.)

For example, a list of IPv4 addresses might look like the following output. The key is “addresses”. Each item in the list is a separate object, separated by braces { }. The objects are two key/value pairs: an IPv4 address (“ip”) and a subnet mask (“netmask”) separated by a comma. The array of objects in the list is also separated by a comma following the closing brace for each object.

```
{
    "addresses": [
        {
            "ip": "172.16.0.2",
            "netmask": "255.255.255.0"
        },
        {
            "ip": "172.16.0.3",
            "netmask": "255.255.255.0"
        },
        {
            "ip": "172.16.0.4",
            "netmask": "255.255.255.0"
        }
    ]
}
```

# YAML Data Format

YAML is another type of human readable data format used by applications for storing, transferring, and reading data. Some of the characteristic of YAML include:

- It is like JSON and is considered a superset of JSON.
- It has a minimalist format making it easy to both read and write.
- It uses indentation to define its structure, without the use of brackets or commas.

# YAML Data Format (Cont.)

```
{
   "ietf-interfaces:interface": {
      "name": "GigabitEthernet2",
      "description": "Wide Area Network",
      "enabled": true,
      "ietf-ip:ipv4": {
         "address": [
            {
               "ip": "172.16.0.2",
               "netmask": "255.255.255.0"
            },
            {
               "ip": "172.16.0.3",
               "netmask": "255.255.255.0"
            },
            {
               "ip": "172.16.0.4",
               "netmask": "255.255.255.0"
            }
         ]
      }
   }
}
```

- IOS output in JSON is to the left. The same data in YAML format is below. It is easier to read.
- Similar to JSON, a YAML object is one or more key value pairs. Key value pairs are separated by a colon without the use of quotation marks. In YAML, a hyphen is used to separate each element in a list.

```
ietf-interfaces:interface:
   name: GigabitEthernet2
   description: Wide Area Network
   enabled: true
   ietf-ip:ipv4:
      address:
         - ip: 172.16.0.2
           netmask: 255.255.255.0
         - ip: 172.16.0.3
           netmask: 255.255.255.0
         - ip: 172.16.0.4
           netmask: 255.255.255.0
```

# XML Data Format

XML is one more type of human readable data format used to store, transfer, and read data by applications. Some of the characteristics of XML include:

- It is like HTML , which is the standardized markup language for creating web pages and web applications.
- It is self-descriptive. It encloses data within a related set of tags: **<tag>data</tag>**
- Unlike HTML, XML uses no predefined tags or document structure.

XML objects are one or more key/value pairs, with the beginning tag used as the name of the key: **<key>value</key>**

# XML Data Format (Cont.)

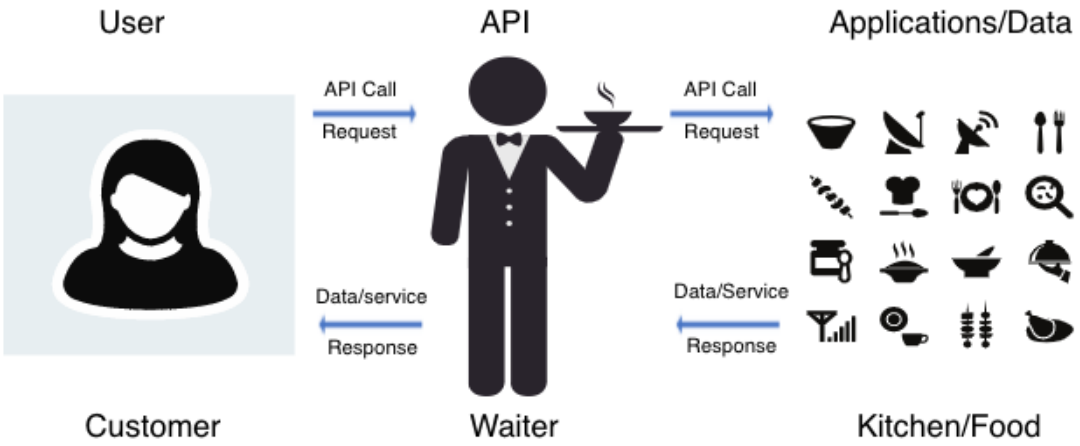
The output shows the same data for GigabitEthernet2 formatted as an XML data structure. Notice how the values are enclosed within the object tags. In this example, each key/value pair is on a separate line and some lines are indented. This is not required but is done for readability. The list uses repeated instances of **<tag></tag>** for each element in the list. The elements within these repeated instances represent one or more key/value pairs.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ietf-interfaces:interface>
   <name>GigabitEthernet2</name>
   <description>Wide Area Network</description>
   <enabled>true</enabled>
   <ietf-ip:ipv4>
      <address>
         <ip>172.16.0.2</ip>
         <netmask>255.255.255.0</netmask>
      </address>
      <address>
         <ip>172.16.0.3</ip>
         <netmask>255.255.255.0</netmask>
      </address>
      <address>
         <ip>172.16.0.4</ip>
         <netmask>255.255.255.0</netmask>
      </address>
   </ietf-ip:ipv4>
</ietf-interfaces:interface>
```

# 14.3 APIs

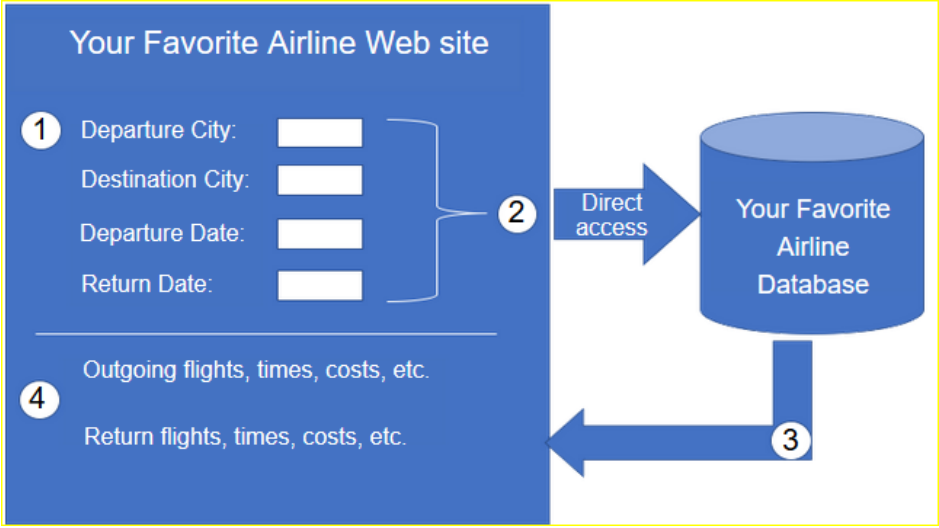
# The API Concept

- An API is software that allows other applications to access its data or services. It is a set of rules describing how one application can interact with another, and the instructions to allow the interaction to occur. The user sends an API request to a server asking for specific information and receives an API response in return from the server along with the requested information.
- An API is similar to a waiter in a restaurant, as shown in the following figure.



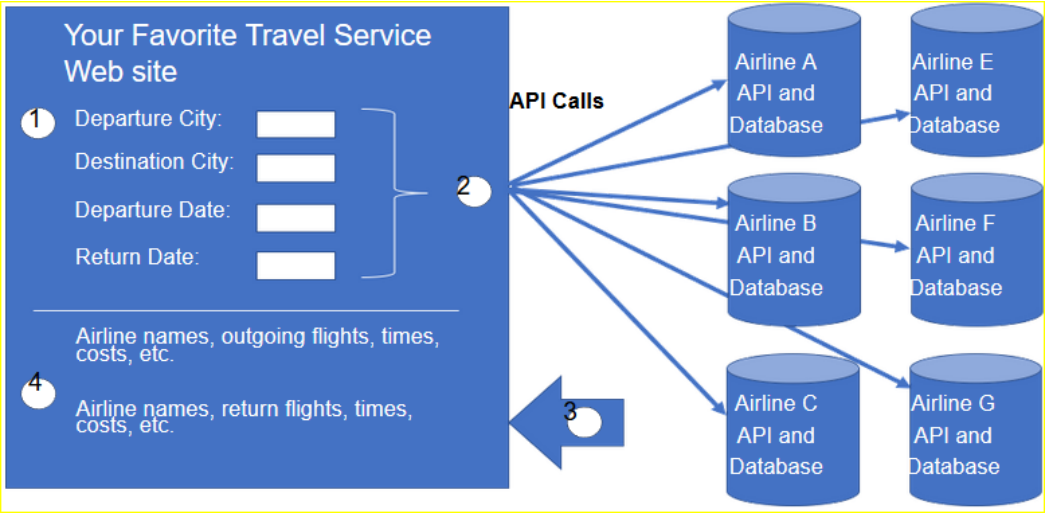
# An API Example

To really understand how APIs can be used to provide data and services, we will look at two options for booking airline reservations. The first option uses the web site of a specific airline. Using the airline's web site, the user enters the information to make a reservation request. The web site interacts directly with the airline's own database and provides the user with information matching the user's request.



# An API Example (Cont.)

A travel site can access this same information, not only from a specific airline but a variety of airlines. In this case, the user enters in similar reservation information. The travel service web site interacts with the various airline databases using APIs provided by each airline. The travel service uses each airline API to request information from that specific airline, and then it displays the information from all the airlines on the its web page. The API acts as a kind of messenger between the requesting application and the application on the server that provides the data or service. The message from the requesting application to the server where the data resides is known as an API call.



# Open, Internal, and Partner APIs

An important consideration when developing an API is the distinction between open, internal, and partner APIs:

- **Open APIs or Public APIs -** These APIs are publicly available and can be used with no restrictions. Because these APIs are public, many API providers require the user to get a free key, or token, prior to using the API. This is to help control the number of API requests they receive and process.
- **Internal or Private APIs -** These are APIs that are used by an organization or company to access data and services for internal use only. An example of an internal API is allowing authorized salespeople access to internal sales data on their mobile devices.
- **Partner APIs -** These are APIs that are used between a company and its business partners or contractors to facilitate business between them. The business partner must have a license or other form of permission to use the API. A travel service using an airline's API is an example of a partner API.

# Types of Web Service APIs

A web service is a service that is available over the internet, using the World Wide Web. There are four types of web service APIs:

- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)
- eXtensible Markup Language-Remote Procedure Call (XML-RPC)
- JavaScript Object Notation-Remote Procedure Call (JSON-RPC)

| Characteristic | SOAP | REST | XML-RPC | JSON-RPC |
|---|---|---|---|---|
| Data Format | XML | JSON, XML, YAML, and others | XML | JSON |
| First released | 1998 | 2000 | 1998 | 2005 |
| Strengths | Well-established | Flexible formatting and most widely used | Well-established, simplicity | Simplicity |

# 14.4 REST

# REST and RESTful API

- Web browsers use HTTP or HTTPS to request (GET) a web page. If successfully requested (HTTP status code 200), web servers respond to GET requests with an HTML coded web page.
- Simply stated, a REST API is an API that works on top of the HTTP protocol. It defines a set of functions developers can use to perform requests and receive responses via HTTP protocol such as GET and POST.
- Conforming to the constraints of the REST architecture is generally referred to as being “RESTful”. An API can be considered “RESTful” if it has the following features:
  - **Client-Server** - The client handles the front end and the server handles the back end. Either can be replaced independently of the other.
  - **Stateless** - No client data is stored on the server between requests. The session state is stored on the client.
  - **Cacheable** - Clients can cache responses to improve performance.

# RESTful Implementation

A RESTful web service is implemented using HTTP. It is a collection of resources with four defined aspects:

- The base Uniform Resource Identifier (URI) for the web service, such as http://example.com/resources.
- The data format supported by the web service. This is often JSON, YAML, or XML but could be any other data format that is a valid hypertext standard.
- The set of operations supported by the web service using HTTP methods.
- The API must be hypertext driven.

RESTful APIs use common HTTP methods including POST, GET, PUT, PATCH and DELETE. As shown in the following table, these correspond to RESTful operations: Create, Read, Update, and Delete (or CRUD).

| HTTP Method | RESTful Operation |
|---|---|
| POST | Create |
| GET | Read |
| PUT/PATCH | Update |
| DELETE | Delete |

# URI, URN, and URL

Web resources and web services such as RESTful APIs are identified using a URI. A URI is a string of characters that identifies a specific network resource. A URI has two specializations:

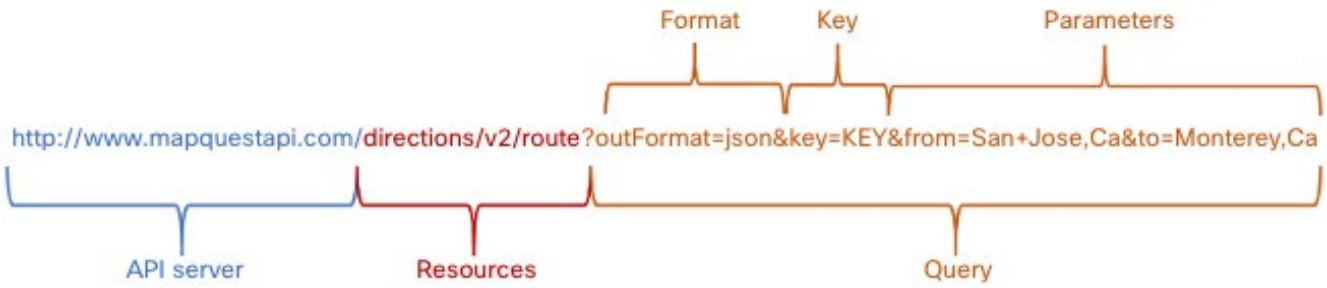
- **Uniform Resource Name (URN)** - identifies only the namespace of the resource (web page, document, image, etc.) without reference to the protocol.
- **Uniform Resource Locator (URL)** - defines the network location of a specific resource. HTTP or HTTPS URLs are typically used with web browsers. Protocols such as FTP, SFTP, SSH, and others can use a URL. A URL using SFTP might look like: sftp://sftp.example.com.

These are the parts of the URI https://www.example.com/author/book.html#page155 :

- **Protocol/scheme** – HTTPS or other protocols such as FTP, SFTP, mailto, and NNTP
- **Hostname** - www.example.com
- **Path and file name** - /author/book.html
- **Fragment** - #page155

# Anatomy of a RESTful Request

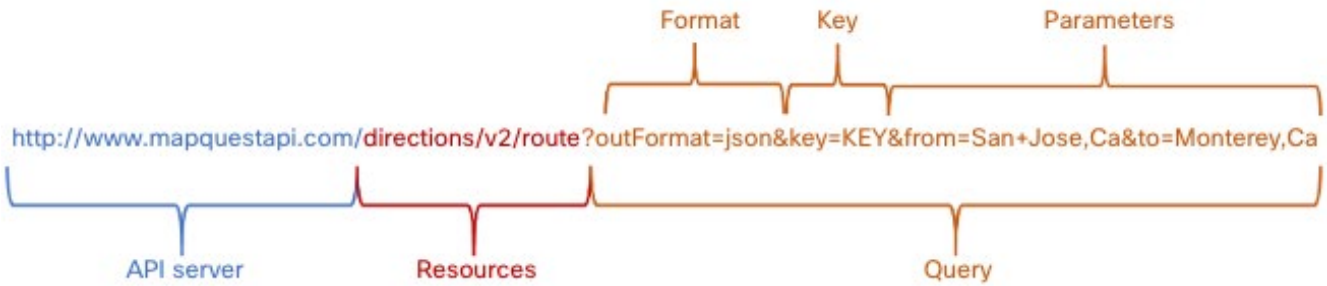
- In a RESTful Web service, a request made to a resource's URI will elicit a response. The response will be a payload typically formatted in JSON, but could be HTML, XML, or some other format. The figure shows the URI for the MapQuest directions API. The API request is for directions from San Jose, California to Monterey, California.



# Anatomy of a RESTful Request (Cont.)

These are the different parts of the API request:

- **API Server** - This is the URL for the server that answers REST requests. In this example it is the MapQuest API server.
- **Resources** - Specifies the API that is being requested. In this example it is the MapQuest directions API.
- **Query** - Specifies the data format and information the client is requesting from the API service. Queries can include:
  - **Format** – This is usually JSON but can be YAML or XML. In this example JSON is requested.
  - **Key** - The key is for authorization, if required. MapQuest requires a key for their directions API. In the above URI, you would need to replace “KEY” with a valid key to submit a valid request.
  - **Parameters** - Parameters are used to send information pertaining to the request. In this example, the query parameters include information about the directions that the API needs so it knows what directions to return: "from=San+Jose,Ca" and "to=Monterey,Ca".



# Anatomy of a RESTful Request (Cont.)

Many RESTful APIs, including public APIs, require a key. The key is used to identify the source of the request. Here are some reasons why an API provider may require a key:

- To authenticate the source to make sure they are authorized to use the API.
- To limit the number of people using the API.
- To limit the number of requests per user.
- To better capture and track the data being requested by users.
- To gather information on the people using the API.

**Note**: The MapQuest API does require a key. Search the internet for the URL to obtain a MapQuest key. Use the search parameters: developer.mapquest. You can also search the internet for the current URL that outlines the MapQuest privacy policy.

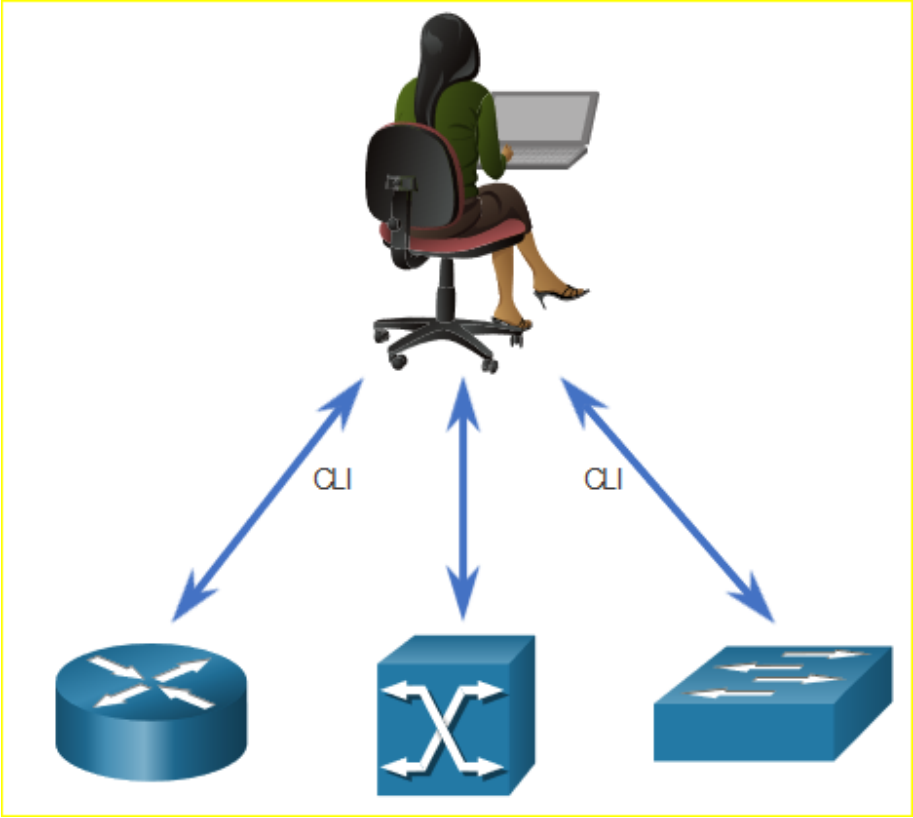
# RESTful API Applications

- Many web sites and applications use APIs to access information and provide service for their customers.
- Some RESTful API requests can be made by typing in the URI from within a web browser. The MapQuest directions API is an example of this. A RESTful API request can also be made in other ways.
  - **Developer Web Site**: Developers often maintain web sites that include information about the API, parameter information, and usage examples. These sites may also allow the user to perform the API request within the developer web page by entering in the parameters and other information.
  - **Postman**: Postman is an application for testing and using REST APIs. It contains everything required for constructing and sending REST API requests, including entering query parameters and keys.
  - **Python**: APIs can also be called from within a Python program. This allows for possible automation, customization, and App integration of the API.
  - **Network Operating Systems**: Using protocols such as NETCONF (NET CONFiguration) and RESTCONF, network operating systems are beginning to provide an alternative method for configuration, monitoring, and management.

# 14.5 Configuration Management Tools

# Traditional Network Configuration

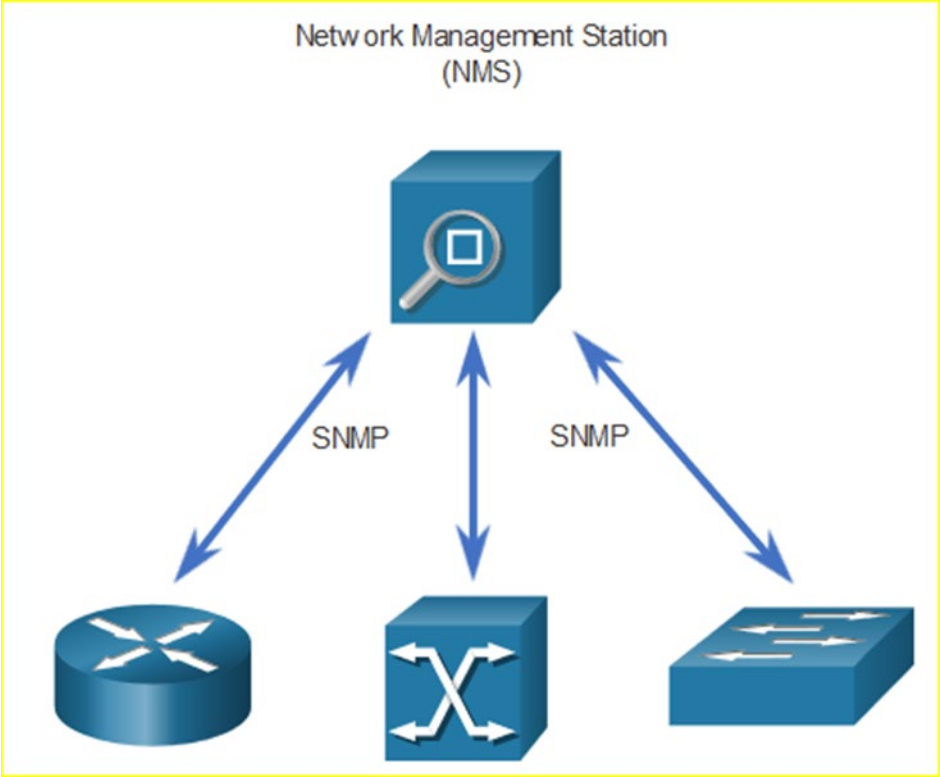
Network devices have traditionally been configured by a network administrator using the CLI. Whenever there is a change or new feature, the necessary configuration commands must be manually entered on all of the appropriate devices. This becomes a major issue on larger networks or with more complex configurations.



# Traditional Network Configuration

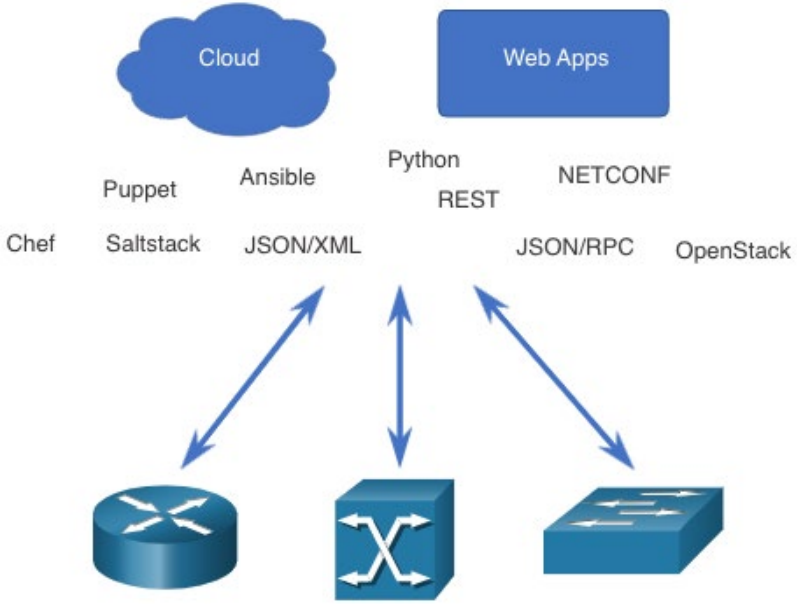
Simple Network Management Protocol (SNMP) lets administrators manage nodes on an IP network. With a network management station (NMS), network administrators use SNMP to monitor and manage network performance, find and solve network problems, and perform queries for statistics. SNMP is not typically used for configuration due to security concerns and difficulty in implementation.

You can also use APIs to automate the deployment and management of network resources. Instead of manually configuring ports, access lists, QoS, and load balancing policies, you can use tools to automate configurations.



# Network Automation

We are rapidly moving away from a world where a network administrator manages a few dozen network devices, to one where they are deploying and managing a great number of complex network devices (both physical and virtual) with the help of software. This transformation is quickly spreading to all places in the network. There are new and different methods for network administrators to automatically monitor, manage, and configure the network. These include protocols and technologies such as REST, Ansible, Puppet, Chef, Python, JSON, XML, and more.



# Configuration Management Tools

Configuration management tools make use of RESTful API requests to automate tasks and can scale across thousands of devices. These are some characteristics of the network that administrators benefit from automating:

- Software and version control
- Device attributes such as names, addressing, and security
- Protocol configurations
- ACL configurations

Configuration management tools typically include automation and orchestration. Automation is when a tool automatically performs a task on a system. Orchestration is the arranging of the automated tasks that results in a coordinate process or workflow.

# Configuration Management Tools (Cont.)

There are several tools available to make configuration management easier:

- Ansible
- Chef
- Puppet
- SaltStack

The goal of all of these tools is to reduce the complexity and time involved in configuring and maintaining a large-scale network infrastructure with hundreds, even thousands of devices. These same tools can benefit smaller networks as well.

# Compare Ansible, Chef, Puppet, and SaltStack

Ansible, Chef, Puppet, and SaltStack all come with API documentation for configuring RESTful API requests. All of them support JSON and YAML as well as other data formats. The following table shows a summary of a comparison of major characteristics of Ansible, Puppet, Chef, and SaltStack configuration management tools.

| Characteristic | Ansible | Chef | Puppet | SaltStack |
|---|---|---|---|---|
| **What programming language?** | Python + YAML | Ruby | Ruby | Python |
| **Agent-based or agentless?** | Agentless | Agent-based | Supports both | Supports both |
| **How are devices managed?** | Any device can be “controller” | Chef Master | Puppet Master | Salt Master |
| **What is created by the tool?** | Playbook | Cookbook | Manifest | Pillar |

# 14.6 IBN and Cisco DNA Center

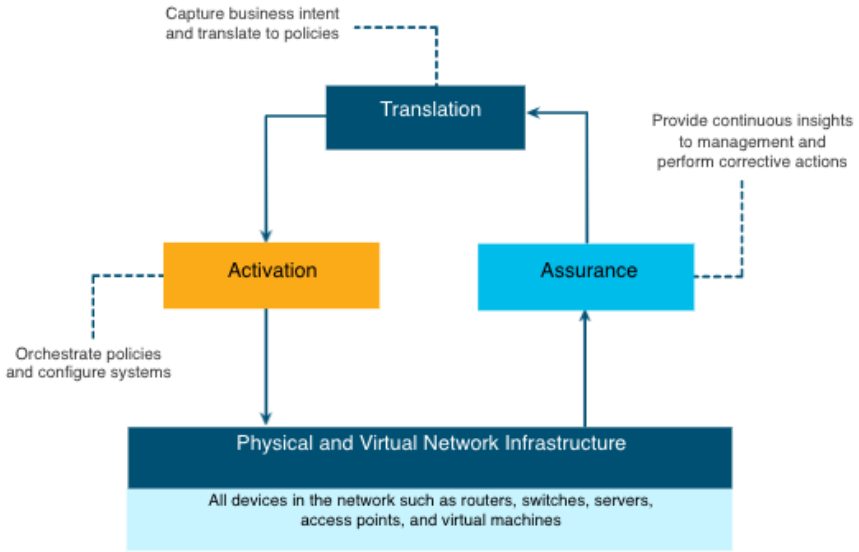
# Intent-Based Networking Overview

- IBN is the emerging industry model for the next generation of networking. IBN builds on Software-Defined Networking (SDN), transforming a hardware-centric and manual approach to designing and operating networks to one that is software-centric and fully automated.
- Business objectives for the network are expressed as intent. IBN captures business intent and uses analytics, machine learning, and automation to align the network continuously and dynamically as business needs change.
- IBN captures and translates business intent into network policies that can be automated and applied consistently across the network.

# Intent-Based Networking Overview (Cont.)

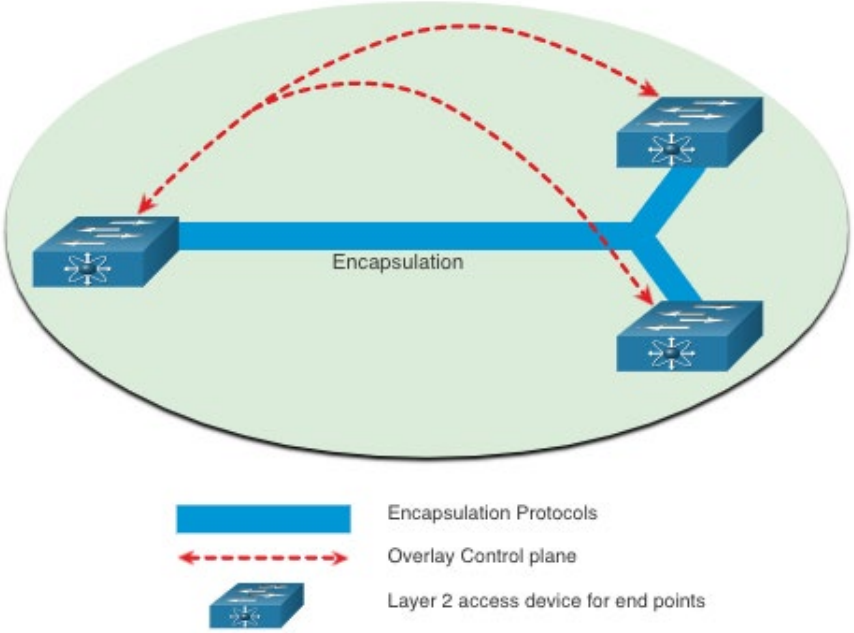
Cisco views IBN as having three essential functions: translation, activation, and assurance. These functions interact with the underlying physical and virtual infrastructure, as shown in the figure.

**Translation** - The translation function enables the network administrator to express the expected networking behavior that will best support the business intent.
**Activation** - The captured intent then needs to be interpreted into policies that can be applied across the network. The activation function installs these policies into the physical and virtual network infrastructure using networkwide automation.
**Assurance** - In order to continuously check that the expressed intent is honored by the network at any point in time, the assurance function maintains a continuous validation-and-verification loop.



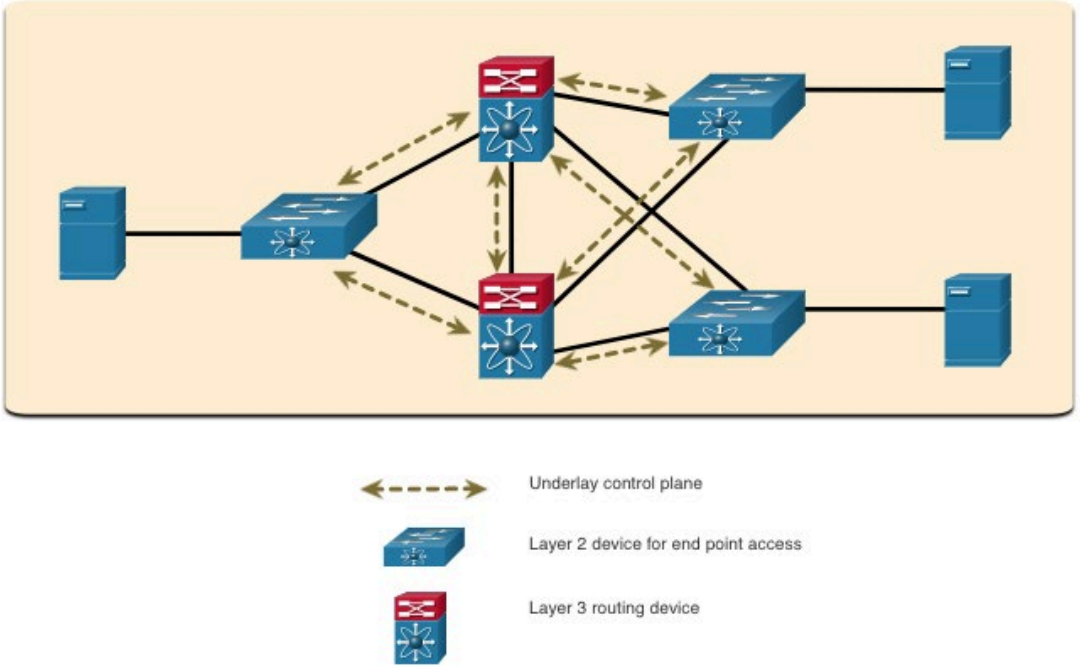
# Network Infrastructure as Fabric

- From the perspective of IBN, the physical and virtual network infrastructure is a fabric; an overlay that represents the logical topology used to virtually connect to devices. The overlay limits the number of devices the network administrator must program and provides services and alternative forwarding methods not controlled by the underlying physical devices.
- The overlay is where encapsulation protocols like IPsec and CAPWAP occur. Using an IBN solution, the network administrator can use policies to specify exactly what happens in the overlay control plane. Notice that how the switches are physically connected is not a concern of the overlay.



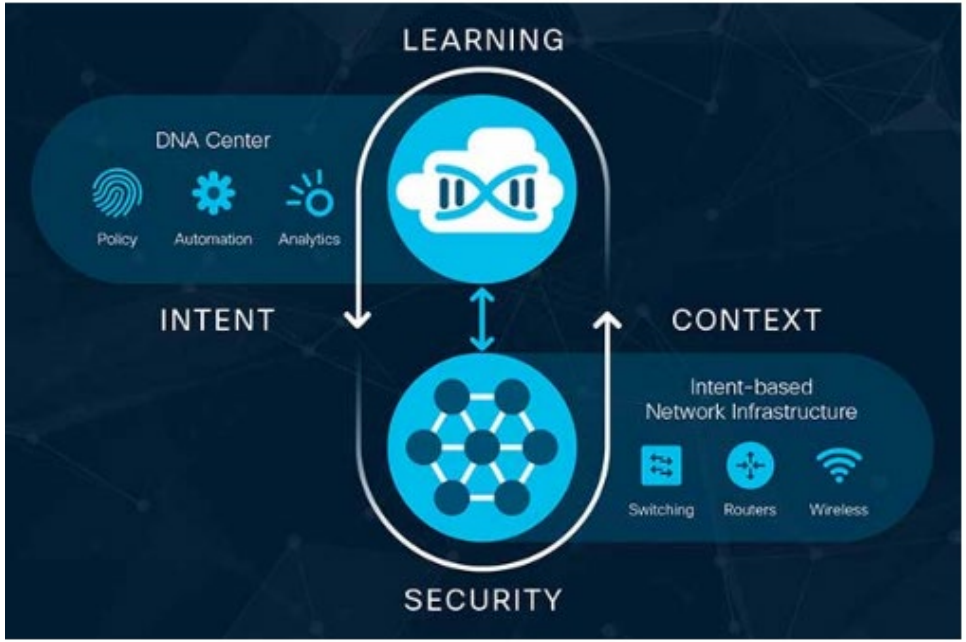
# Network Infrastructure as Fabric (Cont.)

The underlay network is the physical topology that includes all hardware required to meet business objectives. The underlay reveals additional devices and specifies how these devices are connected. End points, such as the servers in the figure, access the network through the Layer 2 devices. The underlay control plane is responsible for simple forwarding tasks.



# Cisco Digital Network Architecture (DNA)

Cisco implements the IBN fabric using Cisco DNA. The business intent is securely deployed into the network infrastructure (the fabric). Cisco DNA then continuously gathers data from a multitude of sources (devices and applications) to provide a rich context of information. This information can then be analyzed to make sure the network is performing securely at its optimal level and in accordance with business intent and network policies.



# Cisco Digital Network Architecture (DNA) (Cont.)

| Cisco DNA Solution | Description | Benefits |
|---|---|---|
| **SD-Access** | •First intent-based enterprise networking solution built using Cisco DNA.<br>•It uses a single network fabric across LAN and WLAN to create a consistent, highly secure user experience.<br>•It segments user, device, and application traffic and automates user-access policies to establish the right policy for any user or device, with any application, across a network. | Enables network access in minutes for any user or device to any application without compromising security. |
| **SD-WAN** | •It uses a secure cloud-delivered architecture to centrally manage WAN connections.<br>•It simplifies and accelerates delivery of secure, flexible and rich WAN services to connect data centers, branches, campuses, and colocation facilities. | •Delivers better user experiences for applications residing on-premise or in the cloud.<br>•Achieve greater agility and cost savings through easier deployments and transport independence. |

# Cisco Digital Network Architecture (DNA) (Cont.)

| Cisco DNA Solution | Description | Benefits |
|---|---|---|
| **Cisco DNA Assurance** | •Used to troubleshoot and increase IT productivity.<br>•It applies advanced analytics and machine learning to improve performance and issue resolution, and predict to assure network performance.<br>•It provides real-time notification for network conditions that require attention. | •Allows you to identify root causes and provides suggested remediation for faster troubleshooting.<br>•The Cisco DNA Center provides an easy-to-use single dashboard with insights and drill-down capabilities.<br>•Machine learning continually improves network intelligence to predict problems before they occur. |
| **Cisco DNA Security** | •Used to provide visibility by using the network as a sensor for real-time analysis and intelligence.<br>•It provides increased granular control to enforce policy and contain threats across the network. | •Reduce risk and protect your organization against threats - even in encrypted traffic.<br>•Gain 360-degree visibility through real-time analytics for deep intelligence across the network.<br>•Lower complexity with end-to-end security. |

# Cisco DNA Center

- Cisco DNA Center is the foundational controller and analytics platform at the heart of Cisco DNA. It supports the expression of intent for multiple use cases, including basic automation capabilities, fabric provisioning, and policy-based segmentation in the enterprise network. Cisco DNA Center is a network management and command center for provisioning and configuring network devices. It is a hardware and software platform providing a ‘single-pane-of-glass’ (single interface) that focuses on assurance, analytics, and automation.
- The DNA Center interface launch page gives you an overall health summary and network snapshot. From here, the network administrator can quickly drill down into areas of interest.

# Cisco DNA Center (Cont.)


At the top, menus provide you access to DNA Center's five main areas. As shown in the figure, these are:

- **Design** - Model your entire network, from sites and buildings to devices and links, both physical and virtual, across campus, branch, WAN, and cloud.
- **Policy** - Use policies to automate and simplify network management, reducing cost and risk while speeding rollout of new and enhanced services.
- **Provision** - Provide new services to users with ease, speed, and security across your enterprise network, regardless of network size and complexity.
- **Assurance** - Use proactive monitoring and insights from the network, devices, and applications to predict problems faster and ensure that policy and configuration changes achieve the business intent and the user experience you want.
- **Platform** - Use APIs to integrate with your preferred IT systems to create end-to-end solutions and add support for multi-vendor devices.

CISCO