CCNA 3 v7.0 Curriculum: Module 14 – Network Automation

itexamanswers.net/ccna-3-v7-0-curriculum-module-14-network-automation.html

April 18, 2020

Contents

14.0. Introduction

14.0.1. Why should I take this module?

Welcome to Network Automation!

Have you set up your home network? A small office network? Imagine doing those tasks for tens of thousands of end devices and thousands of routers, switches, and access points! Did you know that there is software that automates those tasks for an enterprise network? In fact, there is software that can automate the design of an enterprise network. It can automate all of the monitoring, operations and maintenance for your network. Interested? Get started!

14.0.2. What will I learn in this module?

Module Title: Network Automation

Module Objective: Explain how network automation is enabled through RESTful APIs and configuration management tools.

Topic Title	Topic Objective
Automation Overview	Describe automation.
Data Formats	Compare JSON, YAML, and XML data formats.
APIs	Explain how APIs enable computer to computer communications.
REST	Explain how REST enables computer to computer communications.
Configuration Management	Compare the configuration management tools Puppet, Chef, Ansible, and SaltStack.
IBN and Cisco DNA Center	Explain how Cisco DNA center enables intent-based networking.

14.1. Automation Overview

14.1.1 Video – Automation Everywhere

We now see automation everywhere, from self-serve checkouts at stores and automatic building environmental controls, to autonomous cars and planes. How many automated systems do you encounter in a single day?

Click Play in the video to see examples of automation.

14.1.2. The Increase in Automation

Automation is any process that is self-driven, that reduces and potentially eliminates, the need for human intervention.

Automation was once confined to the manufacturing industry. Highly repetitive tasks, such as automobile assembly, were turned over to machines and the modern assembly line was born. Machines excel at repeating the same task without fatigue and without the errors that humans are prone to make in such jobs.

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.

14.1.3. Thinking Devices

Can devices think? Can they learn from their environment? In this context, there are many definitions of the word “think”. One possible definition is the ability to connect a series of related pieces of information together, and then use them to alter a course of action.

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

14.2.1 Video – Data Formats

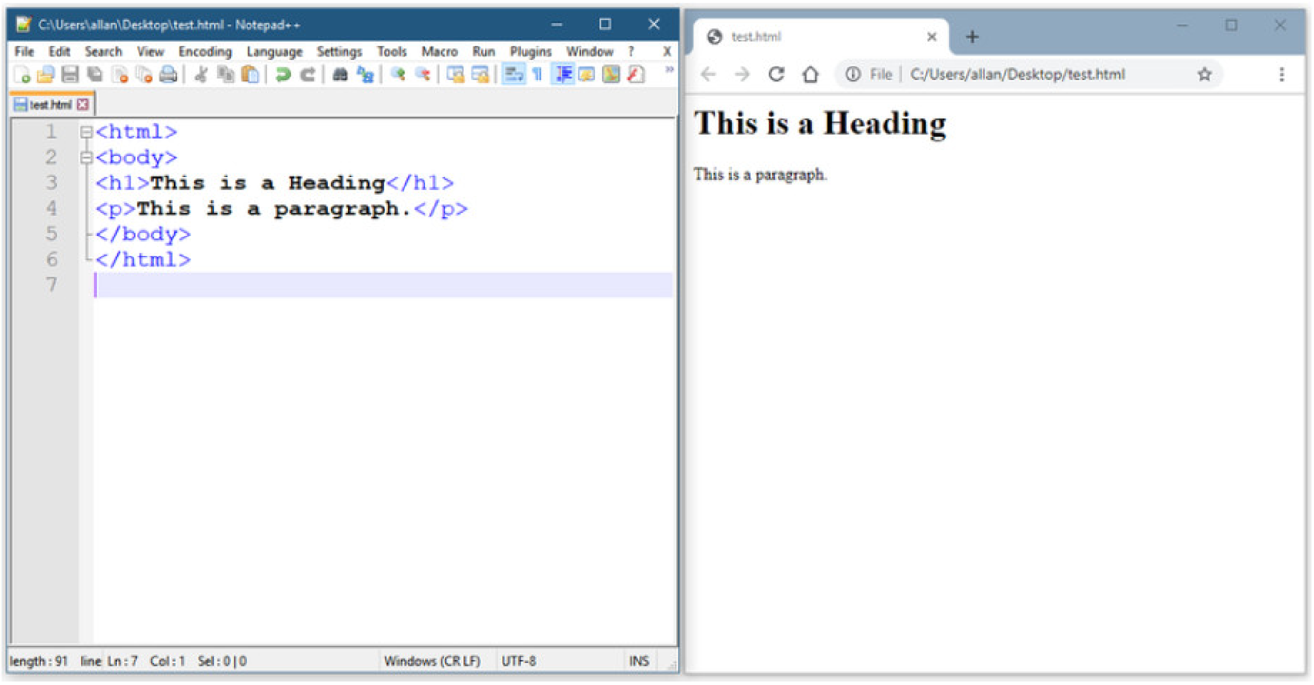
Smart devices are, in fact, tiny computers. For a smart device, such as an actuator, to react to changing conditions, it must be able to receive and interpret information sent to it by another smart device, such as a sensor. These two smart devices must share a common ‘language’ which is called a data format. Shared data formats are also used by other devices in the network.

Click play in the video to learn about data formats.

14.2.2. The Data Formats Concept

When sharing data with people, the possibilities for how to display that information are almost endless. For example, think of how a restaurant might format their menu. It could be text-only, a bulleted list, or photos with captions, or just photos. These are all different ways in which the restaurant can format the data that makes up the menu. A well-designed form is dictated by what makes the information the easiest for the intended audience to understand. This same principle applies to shared data between computers. A computer must put the data into a format that another computer can understand.

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, as shown the figure.


C:\Users\allan\Desktop\test.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
test.html
length : 91 line Ln : 7 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS
test.html
File | C:/Users/allan/Desktop/test.html

```
<html>
<body>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

This is a Heading

This is a paragraph.

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.

14.2.3. 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.

Search the internet for “open notify ISS location now” to find a web site that tracks the current location of the International Space Station. At this web site you can see how data formats are used and some of the similarities between them. This web site includes a link for a simple Application Programming Interface (API) call to a server, which returns the current latitude and longitude of the space station along with a UNIX timestamp. The following example shows the information returned by the server using JavaScript Object Notation (JSON). The information is displayed in a raw format. This can make it difficult to understand the structure of the data.

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

Search the internet to find the “JSONView” browser extension or any extension that will allow you to view JSON in a more readable format. Data objects are displayed in key/value pairs. The following output shows this same output using JSONView. The key/value pairs are much easier to interpret. In the example below, you can see the key **latitude** and its value **25.9990.**

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

Note: JSONView may remove the quotation marks from the key. Quotation marks are required when coding JSON key/value pairs.

14.2.4. Compare Data Formats

To see this same data formatted as XML or YAML, search the internet for a JSON conversion tool. At this point it is not important to understand the details of each data format, but notice how each data format makes use of syntax and how the key/value pairs are represented.

JSON Format

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

YAML Format

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

XML 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>
```

14.2.5. 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.

The following output shows an example of partial IOS output from a **show interface GigabitEthernet0/0/0** command on a router.

IOS Router Output

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

This same information can be represented 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.

JSON Output

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

14.2.6. 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.

In 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.
- Whitespace is not significant.

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.

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.

JSON List of IPv4 Addresses

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

14.2.7. 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.

For example, look at this JSON output for a Gigabit Ethernet 2 interface.

JSON for GigabitEthernet2

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

That same data in YAML format 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. This is shown for the three IPv4 addresses in the following output.

YAML for GigabitEthernet2

```
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
```

14.2.8. 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>**

The following 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 for GigabitEthernet2

```
<?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

14.3.1 Video – APIs

Data formats shared between smart devices often use an Application Programming Interface (API). As you will learn in this topic, an API is software that allows other applications to access its data or services.

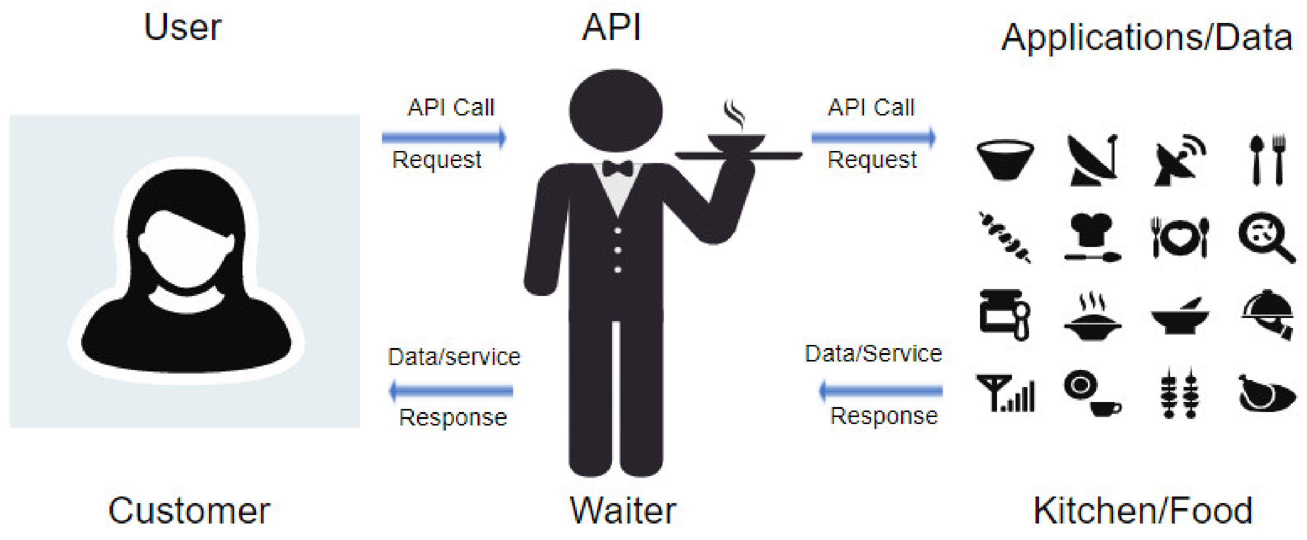
Click play in the video to learn about APIs.

14.3.2. The API Concept

APIs are found almost everywhere. Amazon Web Services, Facebook, and home automation devices such as thermostats, refrigerators, and wireless lighting systems, all use APIs. They are also used for building programmable network automation.

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.

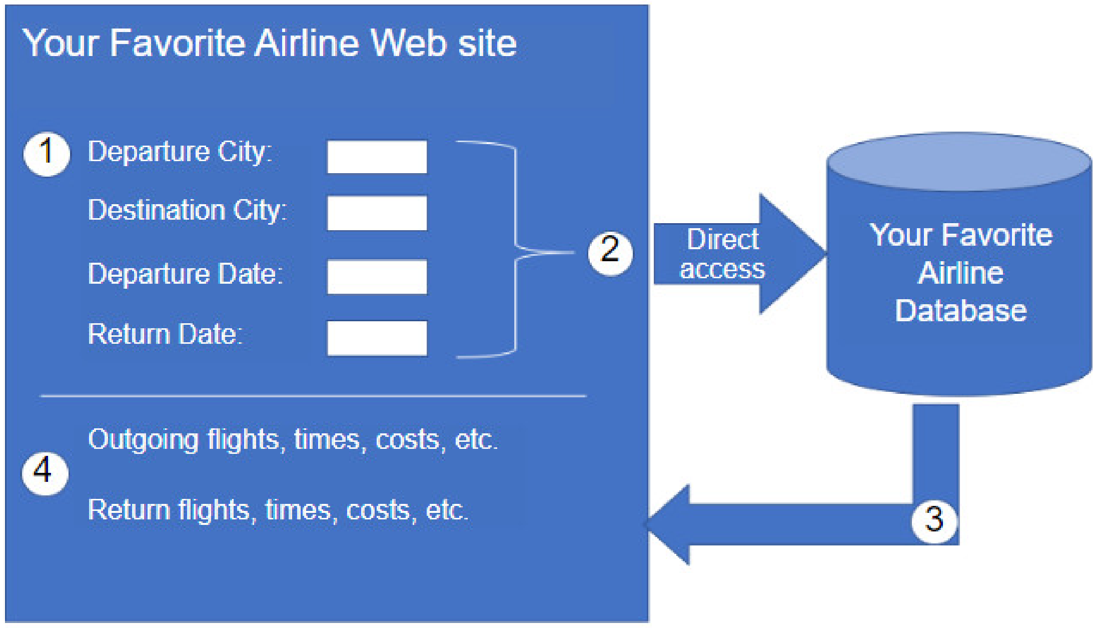
An API is similar to a waiter in a restaurant, as shown in the following figure. A customer in a restaurant would like to have some food delivered to the table. The food is in the kitchen where it is cooked and prepared. The waiter is the messenger, similar to an API. The waiter (the API) is the person who takes the customer's order (the request) and tells the kitchen what to do. When the food is ready, the waiter will then deliver the food (the response) back to the customer.



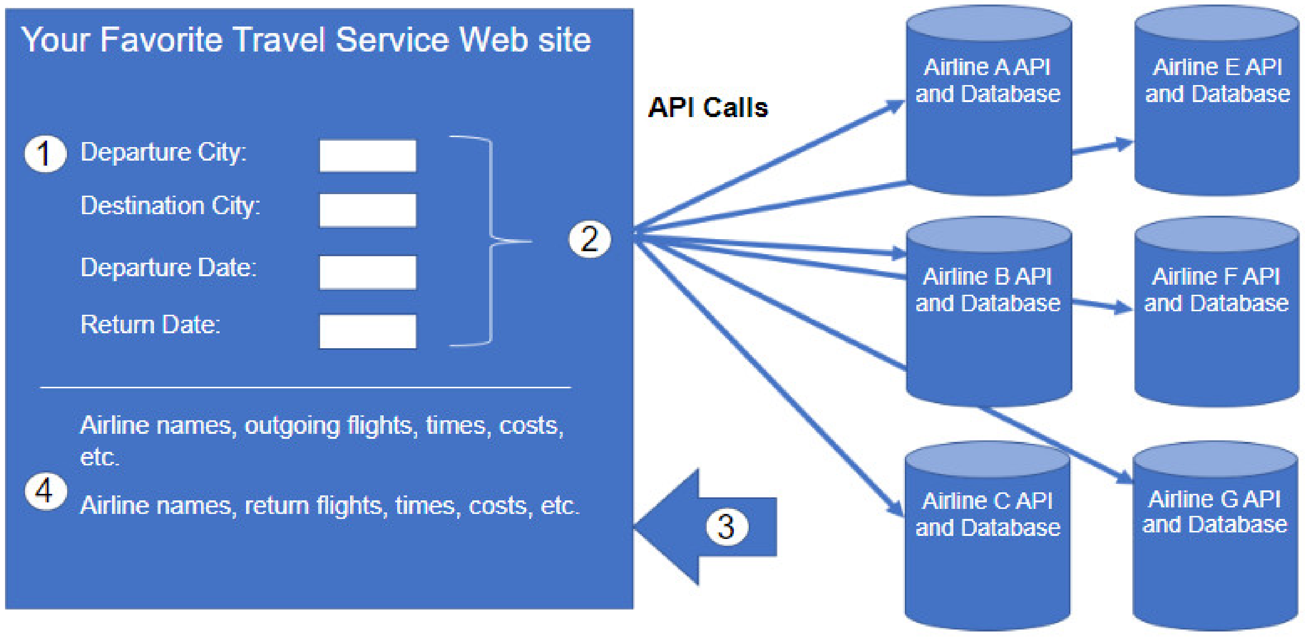
Previously, you saw an API request to a server which returned the current latitude and longitude of the International Space Station. This was an API that Open Notify provides to access data from a web browser at National Aeronautics and Space Administration (NASA).

14.3.3. 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, as shown in the figure. 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.


Your Favorite Airline Web site
1
Departure City:
Destination City:
Departure Date:
Return Date:
2
Direct access
Your Favorite Airline Database
3
4
Outgoing flights, times, costs, etc.
Return flights, times, costs, etc.

Instead of using an individual airline web site which has direct access to its own information, there is a second option. Users can use a travel site to 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, as shown in the figure.

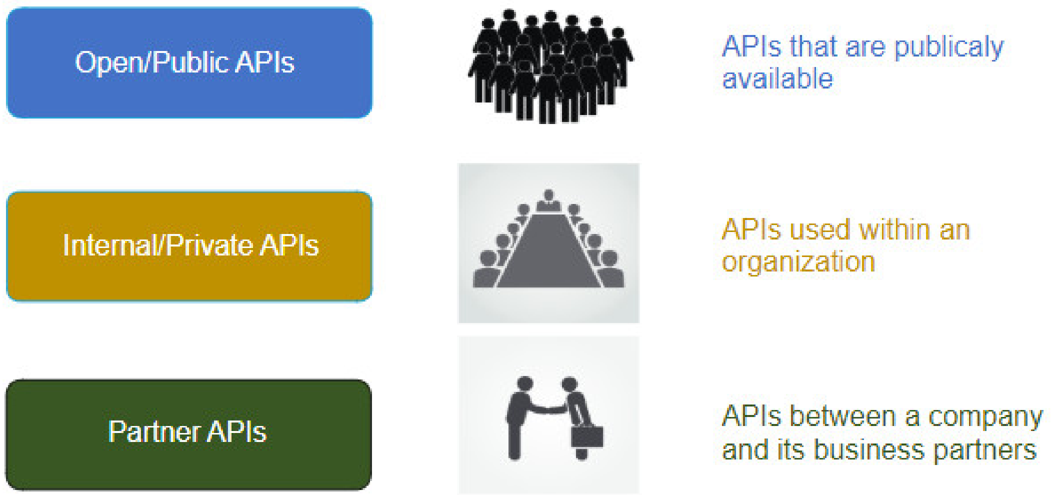

Your Favorite Travel Service Web site
API Calls
1
Departure City:
Destination City:
Departure Date:
Return Date:
2
Airline A API and Database
Airline E API and Database
Airline B API and Database
Airline F API and Database
Airline C API and Database
Airline G API and Database
3
4
Airline names, outgoing flights, times, costs, etc.
Airline names, return flights, times, costs, etc.

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.

14.3.4. 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. The International Space Station API is an example of a Public API. Because these APIs are public, many API providers, such as Google Maps, 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. Search the internet for a list of public APIs.
- **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.


Open/Public APIs
APIs that are publicaly available
Internal/Private APIs
APIs used within an organization
Partner APIs
APIs between a company and its business partners

14.3.5. 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

SOAP is a messaging protocol for exchanging XML-structured information, most often over HTTP or Simple Mail Transfer Protocol (SMTP). Designed by Microsoft in 1998, SOAP APIs are considered slow to parse, complex, and rigid.

This led to the development of a simpler REST API framework which does not require XML. REST uses HTTP, is less verbose, and is easier to use than SOAP. REST refers to the style of software architecture and has become popular due to its performance, scalability, simplicity, and reliability.

REST is the most widely used web service API, accounting for over 80% of all the API types used. REST will be further discussed in this module.

RPC is when one system requests that another system executes some code and returns the information. This is done without having to understand the details of the network. This works much like a REST API but there are differences dealing with formatting and flexibility. XML-RPC is a protocol developed prior to SOAP, and later evolved into what became SOAP. JSON-RPC is a very simple protocol and similar to XML-RPC.

14.4. REST

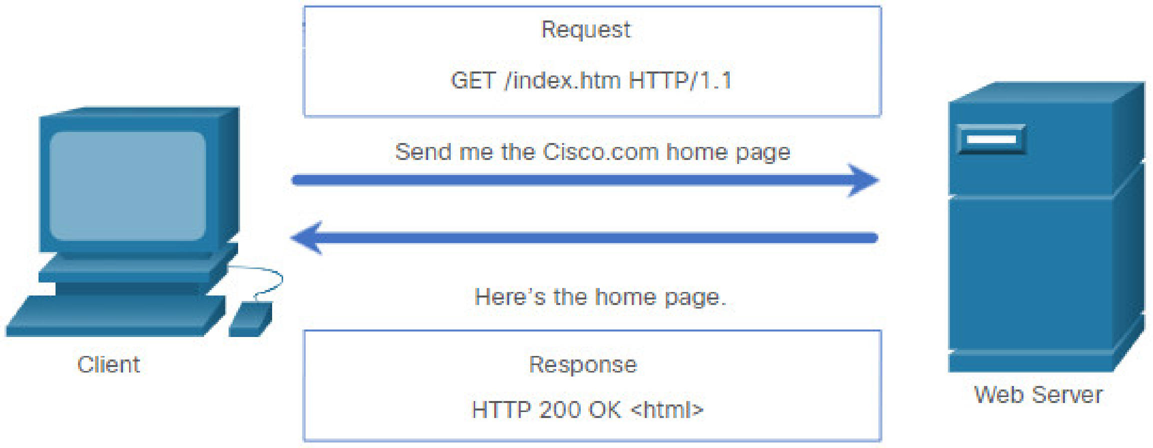
14.4.1 Video – REST

As you have just learned, REST is currently the most widely used API. This topic covers REST in more detail.

Click play in the video to learn about more about REST.

14.4.2. 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, as shown in the figure.



REST is an architectural style for designing web service applications. It refers to a style of web architecture that has many underlying characteristics and governs the behavior of clients and servers. 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.

14.4.3. RESTful Implementation

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

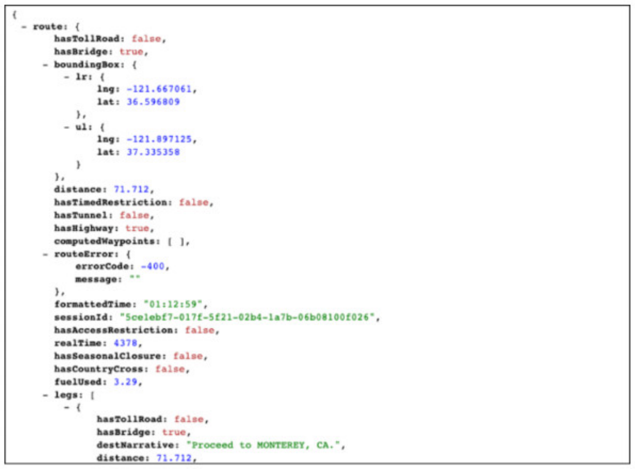
- The base Uniform Resource Identifier (URI) for the web service, such as ht tp://example.com/resources.
- The data format supported by the web service. This is often JSON, YAML, or XML but could be any other data format that is a valid hypertext standard.
- The set of operations supported by the web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

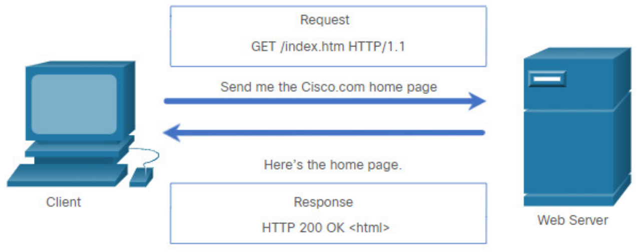
HTTP Method	RESTful Operation

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

In the figure, the HTTP request asks for JSON-formatted data. If the request is successfully constructed according to the API documentation, the server will respond with JSON data. This JSON data can be used by a client's web application to display the data. For example, a smartphone mapping app show the location of San Jose, California.



```
{
 - route: {
      hasTollRoad: false,
      hasBridge: true,
    - boundingBox: {
       - lr: {
            lng: -121.667061,
            lat: 36.596809
         },
       - ul: {
            lng: -121.897125,
            lat: 37.335358
         }
      },
      distance: 71.712,
      hasTimedRestriction: false,
      hasTunnel: false,
      hasHighway: true,
      computedWaypoints: [ ],
    - routeError: {
         errorCode: -400,
         message: ""
      },
      formattedTime: "01:12:59",
      sessionId: "5celebf7-017f-5f21-02b4-1a7b-06b08100f026",
      hasAccessRestriction: false,
      realTime: 4378,
      hasSeasonalClosure: false,
      hasCountryCross: false,
      fuelUsed: 3.29,
    - legs: [
       - {
            hasTollRoad: false,
            hasBridge: true,
            destNarrative: "Proceed to MONTEREY, CA.",
            distance: 71.712,
```



14.4.4. 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. As shown in the figure, a URI has two specializations:

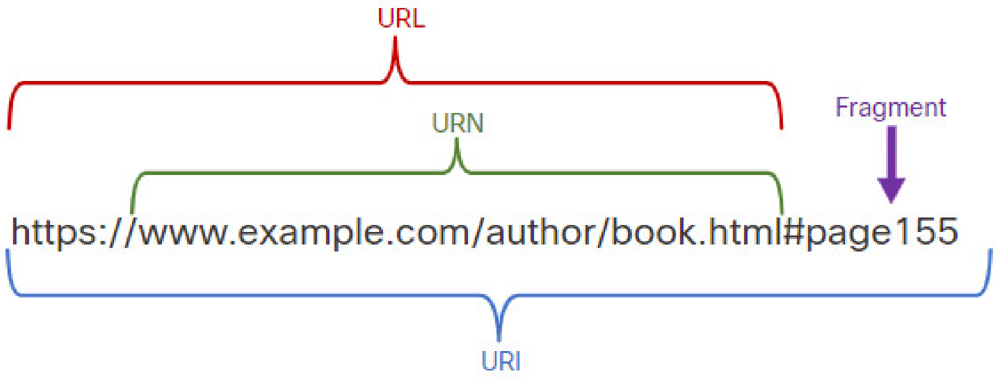
- **Uniform Resource Name (URN)** – identifies only the namespace of the resource (web page, document, image, etc.) without reference to the protocol.
- **Uniform Resource Locator (URL)** – defines the network location of a specific resource on the network. HTTP or HTTPS URLs are typically used with web browsers. Other 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 a URI, as shown in the figure:

- **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

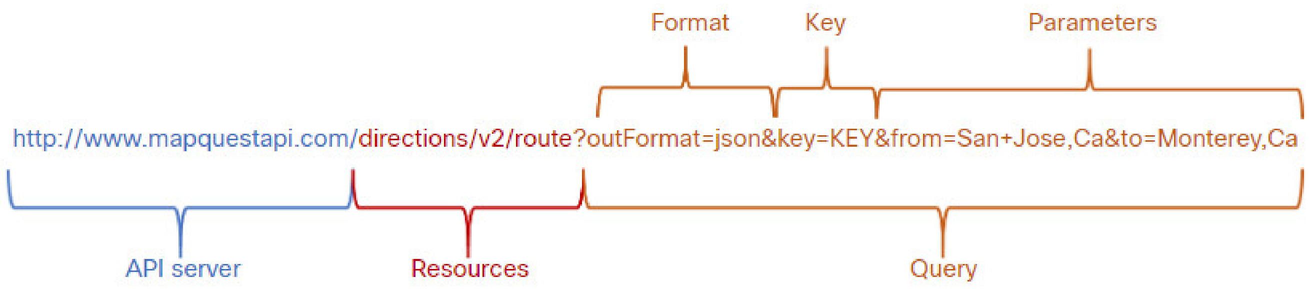
Parts of a URI



14.4.5. Anatomy of a RESTful Request

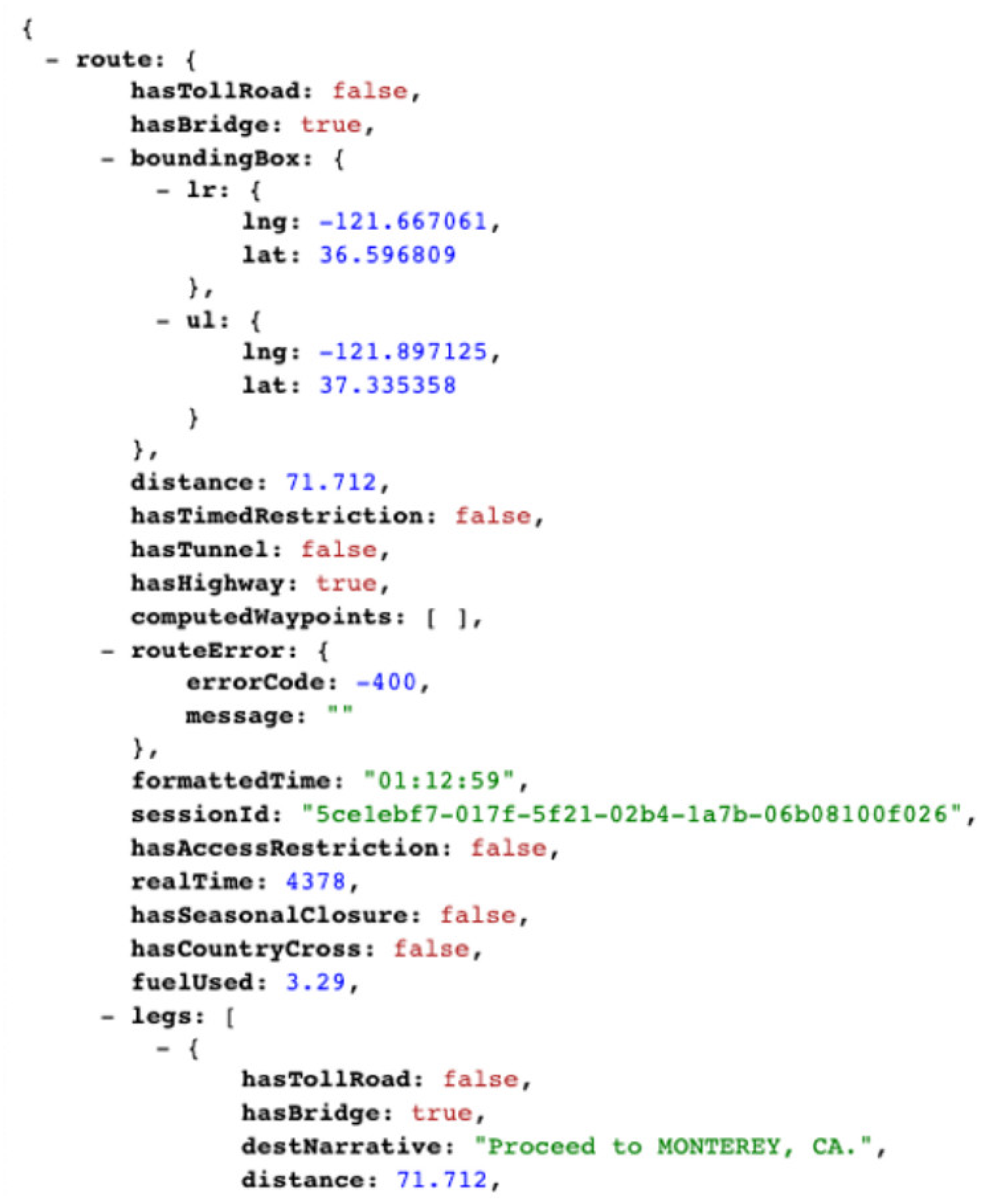
In a RESTful Web service, a request made to a resource's URI will elicit a response. The response will be a payload typically formatted in JSON, but could be HTML, XML, or some other format. The figure shows the URI for the MapQuest directions API. The API request is for directions from San Jose, California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

Parts of an API Request


Format
Key
Parameters
http://www.mapquestapi.com/directions/v2/route?outFormat=json&key=KEY&from=San+Jose,Ca&to=Monterey,Ca
API server
Resources
Query

The figure shows part of the API response. In this example it is the MapQuest directions from San Jose to Monterey in JSON format.

Partial JSON Payload Received from an API Request



```
{
  - route: {
        hasTollRoad: false,
        hasBridge: true,
      - boundingBox: {
          - lr: {
                lng: -121.667061,
                lat: 36.596809
            },
          - ul: {
                lng: -121.897125,
                lat: 37.335358
            }
        },
        distance: 71.712,
        hasTimedRestriction: false,
        hasTunnel: false,
        hasHighway: true,
        computedWaypoints: [ ],
      - routeError: {
            errorCode: -400,
            message: ""
        },
        formattedTime: "01:12:59",
        sessionId: "5ce1ebf7-017f-5f21-02b4-1a7b-06b08100f026",
        hasAccessRestriction: false,
        realTime: 4378,
        hasSeasonalClosure: false,
        hasCountryCross: false,
        fuelUsed: 3.29,
      - legs: [
          - {
                hasTollRoad: false,
                hasBridge: true,
                destNarrative: "Proceed to MONTEREY, CA.",
                distance: 71.712,
```

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".

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: If you wish to use the MapQuest API, the 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.

14.4.6. RESTful API Applications

Many web sites and applications use APIs to access information and provide service for their customers. For example, when using a travel service web site, the travel service uses the API of various airlines to provide the user with airline, hotel and other information.

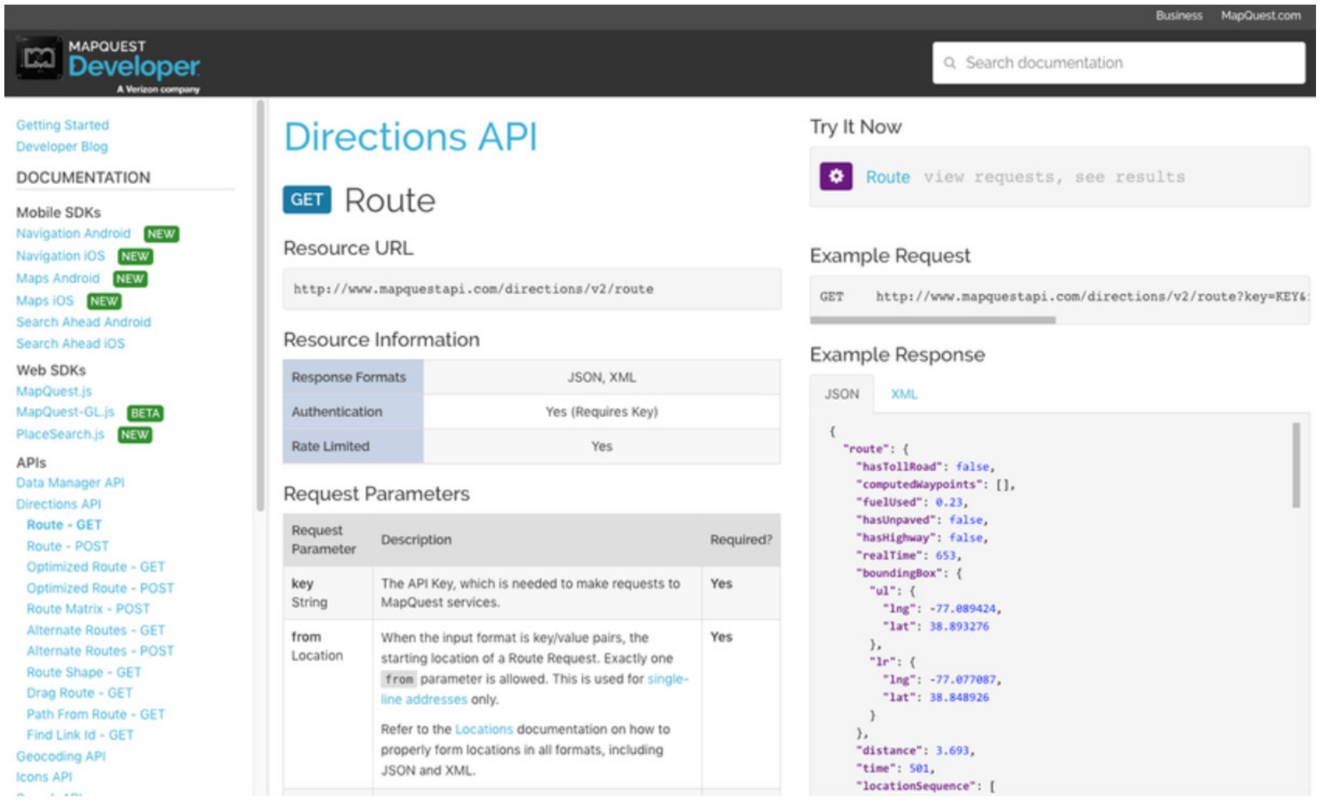
Some RESTful API requests can be made by typing in the URI from within a web browser. The MapQuest directions API is an example of this. A RESTful API request can also be made in other ways.

Click each API application scenario below for more information.

- Developer Web Site
- Postman
- Python
- Network Operating Systems

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. The following figure shows an example of the MapQuest Directions API web page.



Directions API

GET Route

Resource URL

```
http://www.mapquestapi.com/directions/v2/route
```

Resource Information

Response Formats	JSON, XML
Authentication	Yes (Requires Key)
Rate Limited	Yes

Request Parameters

Request Parameter	Description	Required?
key String	The API Key, which is needed to make requests to MapQuest services.	Yes
from Location	When the input format is key/value pairs, the starting location of a Route Request. Exactly one `from` parameter is allowed. This is used for single-line addresses only. Refer to the Locations documentation on how to properly form locations in all formats, including JSON and XML.	Yes

Try It Now

Route `view requests, see results`

Example Request

```
GET    http://www.mapquestapi.com/directions/v2/route?key=KEY&
```

Example Response

JSON XML

```
{
  "route": {
    "hasTollRoad": false,
    "computedWaypoints": [],
    "fuelUsed": 0.23,
    "hasUnpaved": false,
    "hasHighway": false,
    "realTime": 653,
    "boundingBox": {
      "ul": {
        "lng": -77.089424,
        "lat": 38.893276
      },
      "lr": {
        "lng": -77.077087,
        "lat": 38.848926
      }
    },
    "distance": 3.693,
    "time": 501,
    "locationSequence": [
```

14.5. Configuration Management Tools

14.5.1 Video – Configuration Management Tools

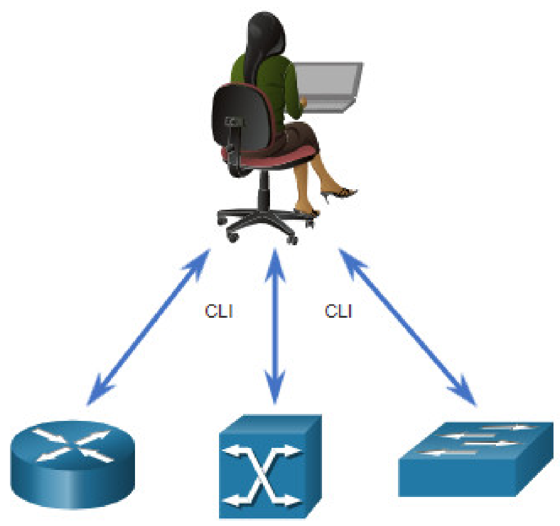
As mentioned in the introduction to this module, setting up a network can be very time consuming. Configuration management tools can help you to automate the configuration of routers, switches, firewalls and many other aspects of your network.

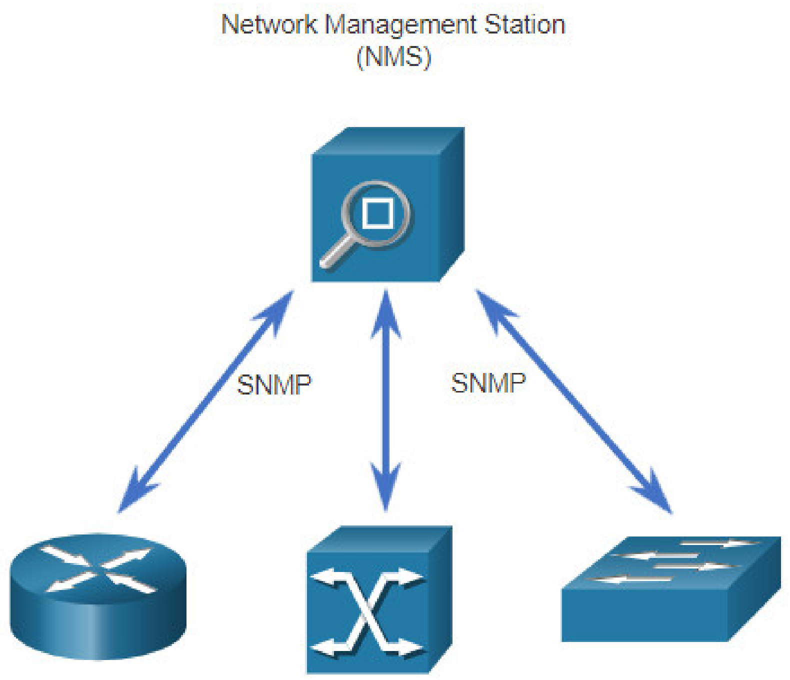
Click play in the video to learn about configuration management tools.

14.5.2. Traditional Network Configuration

Network devices such as router, switches, and firewalls have traditionally been configured by a network administrator using the CLI, as shown in the figure. Whenever there is a change or new feature, the necessary configuration commands must be manually entered on all of the appropriate devices. In many cases, this is not only time-consuming, but can also be prone to errors. This becomes a major issue on larger networks or with more complex configurations.

Simple Network Management Protocol (SNMP) was developed to allow administrators to manage nodes such as servers, workstations, routers, switches, and security appliances, on an IP network. Using a network management station (NMS), shown in the following figure, SNMP enables network administrators to monitor and manage network performance, find and solve network problems, and perform queries for statistics. SNMP works reasonably well for device monitoring. However, it is not typically used for configuration due to security concerns and difficulty in implementation. Although SNMP is widely available, it cannot serve as an automation tool for today's networks.

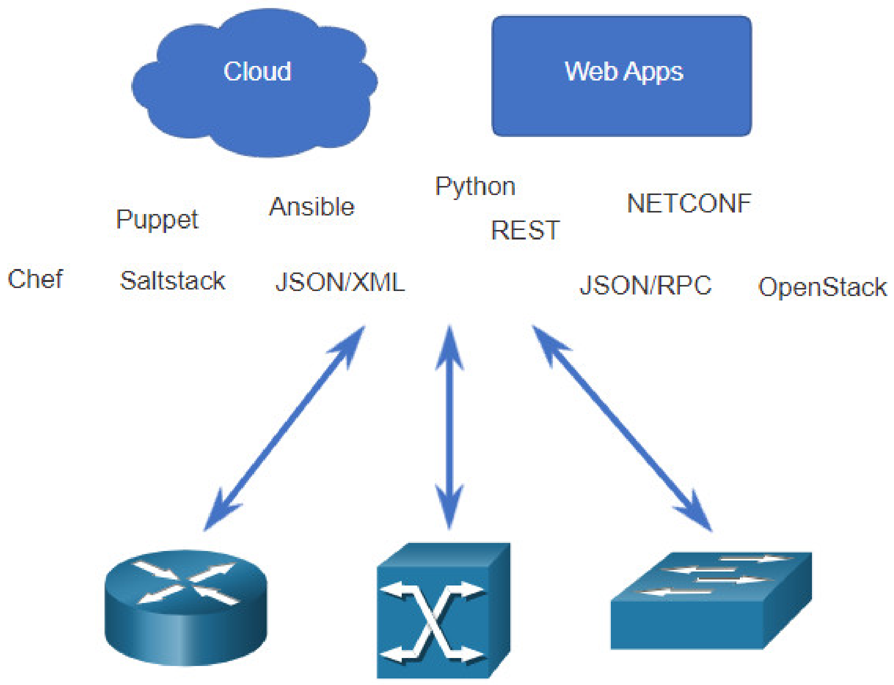




You can also use APIs to automate the deployment and management of network resources. Instead of the network administrator manually configuring ports, access lists, quality of service (QoS), and load balancing policies, they can use tools to automate configurations. These tools hook into network APIs to automate routine network provisioning tasks, enabling the administrator to select and deploy the network services they need. This can significantly reduce many repetitive and mundane tasks to free up time for network administrators to work on more important things.

14.5.3. 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 hundreds, thousands, and even tens of thousands of complex network devices (both physical and virtual) with the help of software. This transformation is quickly spreading from its beginnings in the data center, to all places in the network. There are new and different methods for network operators to automatically monitor, manage, and configure the network. As shown in the figure, these include protocols and technologies such as REST, Ansible, Puppet, Chef, Python, JSON, XML, and more.



14.5.4. Configuration Management Tools

Configuration management tools make use of RESTful API requests to automate tasks and can scale across thousands of devices. Configuration management tools maintain the characteristics of a system, or network, for consistency. 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. This might be configuring an interface or deploying a VLAN. Orchestration is the process of how all these automated activities need to happen, such as the order in which they must be done, what must be completed before another task is begun, etc. Orchestration is the arranging of the automated tasks that results in a coordinate process or workflow.

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.

14.5.5. 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

- **What programming language?** – Ansible and SaltStack are both built on Python whereas Puppet and Chef are built on Ruby. Similar to Python, Ruby is an open-source programming language that is cross-platform. However, Ruby is typically considered a more difficult language to learn than Python.

- **Agent-based or agentless?** – Configuration management is either agent-based or agentless. Agent-based configuration management is “pull-based”, meaning the agent on the managed device periodically connects with the master for its configuration information. Changes are done on the master and pulled down and executed by the device. Agentless configuration management is “push-based.” A configuration script is run on the master. The master connects to the device and executes the tasks in the script. Of the four configuration tools in the table, only Ansible is agentless.
- **How are devices managed?** – This lies with a device called the Master in Puppet, Chef, and SaltStack. However, because Ansible is agentless, any computer can be the controller.
- **What is created by the tool?** – Network administrators use configuration management tools to create a set of instructions to be executed. Each tool has its own name for these instructions: Playbook, Cookbook, Manifest, and Pillar. Common to each of this is specification of a policy or a configuration that is to be applied to devices. Each device type might have its own policy. For example, all Linux servers might get the same basic configuration and security policy.

14.6. IBN and Cisco DNA Center

14.6.1 Video – Intent-Based Networking

You have learned of the many tools and software that can help you automate your network. Intent-Based Networking (IBN) and Cisco Digital Network Architecture (DNA) Center can help you bring it all together to create an automated network.

Click Play in the figure to view a video by Cisco’s John Apostolopoulos and Anand Oswal explaining how artificial intelligence and intent-based networking (IBN) can improve networks.

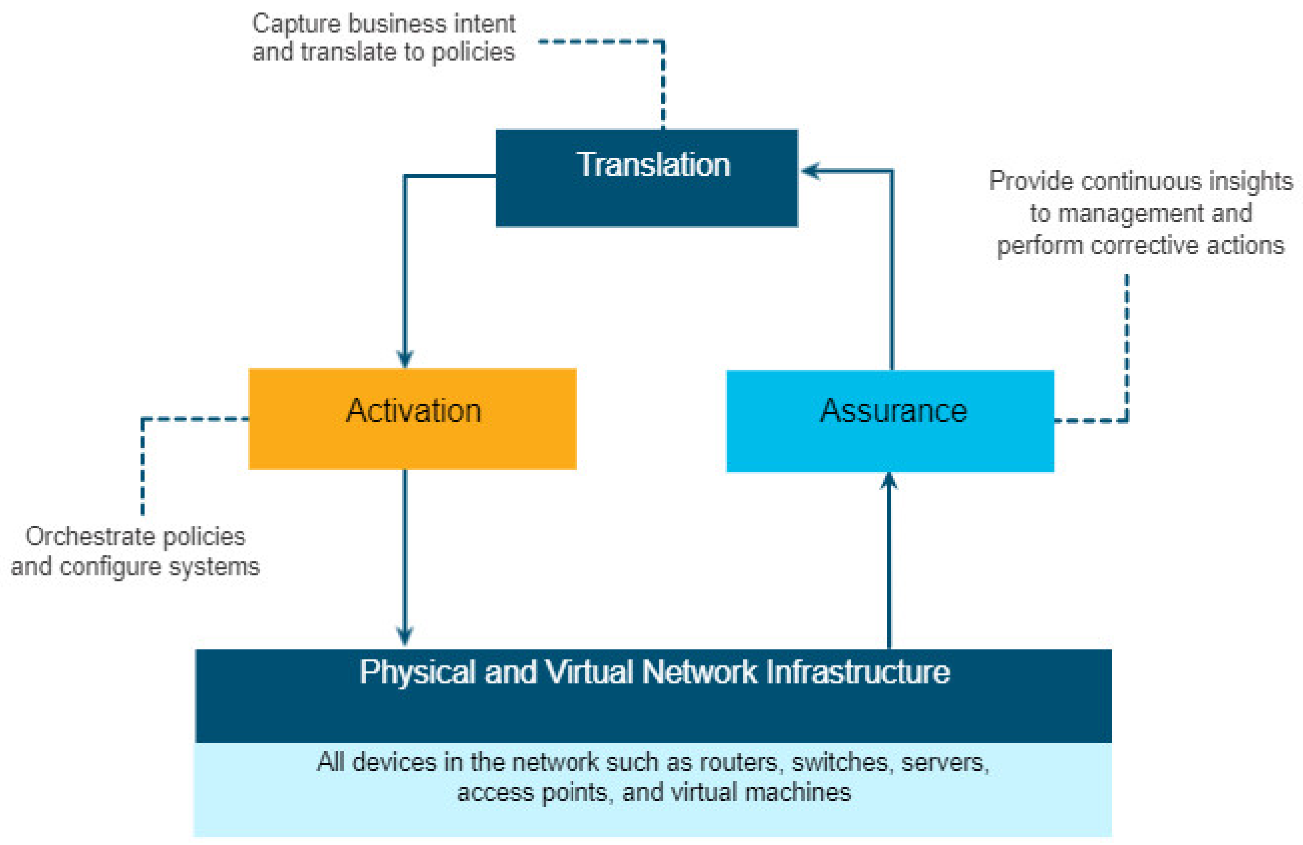
14.6.2. Intent-Based Networking Overview

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.

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.


Capture business intent
and translate to policies
Translation
Provide continuous insights
to management and
perform corrective actions
Activation
Assurance
Orchestrate policies
and configure systems
Physical and Virtual Network Infrastructure
All devices in the network such as routers, switches, servers,
access points, and virtual machines

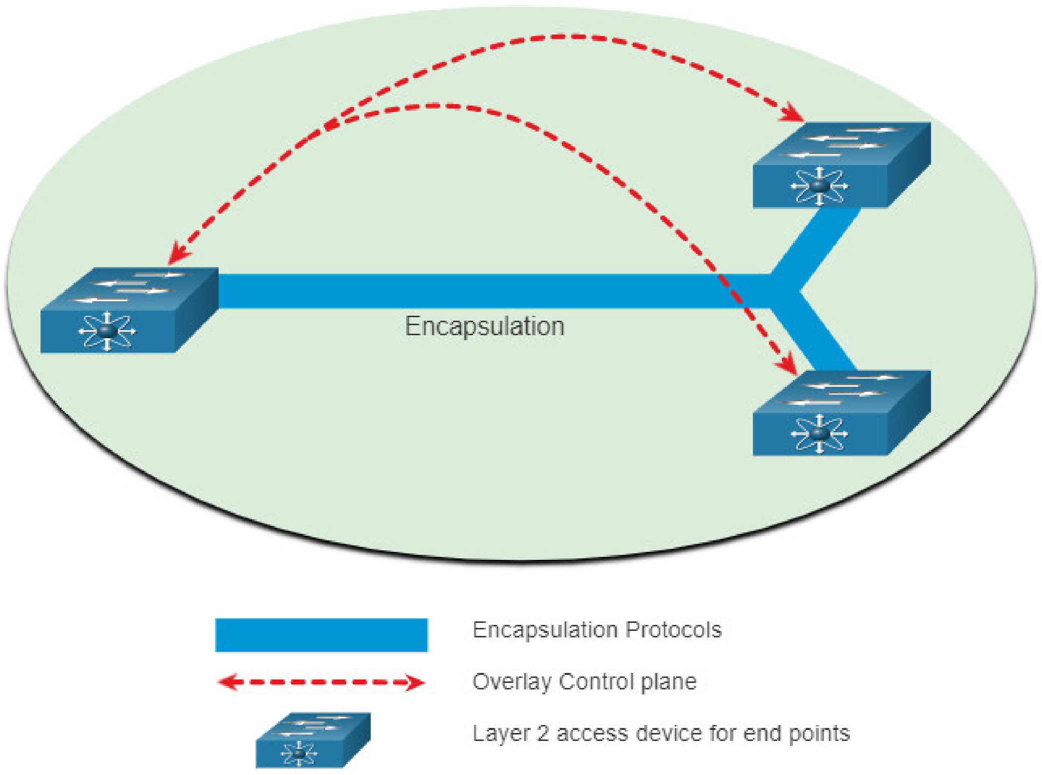
- **Translation** – The translation function enables the network administrator to express the expected networking behavior that will best support the business intent.
- **Activation** – The captured intent then needs to be interpreted into policies that can be applied across the network. The activation 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.

14.6.3. Network Infrastructure as Fabric

From the perspective of IBN, the physical and virtual network infrastructure is a fabric. Fabric is a term used to describe an overlay that represents the logical topology used to virtually connect to devices, as shown in the figure. The overlay limits the number of devices the network administrator must program. It also provides services and alternative forwarding methods not controlled by the underlying physical devices. For example, the overlay is where encapsulation protocols like IP security (IPsec) and Control and

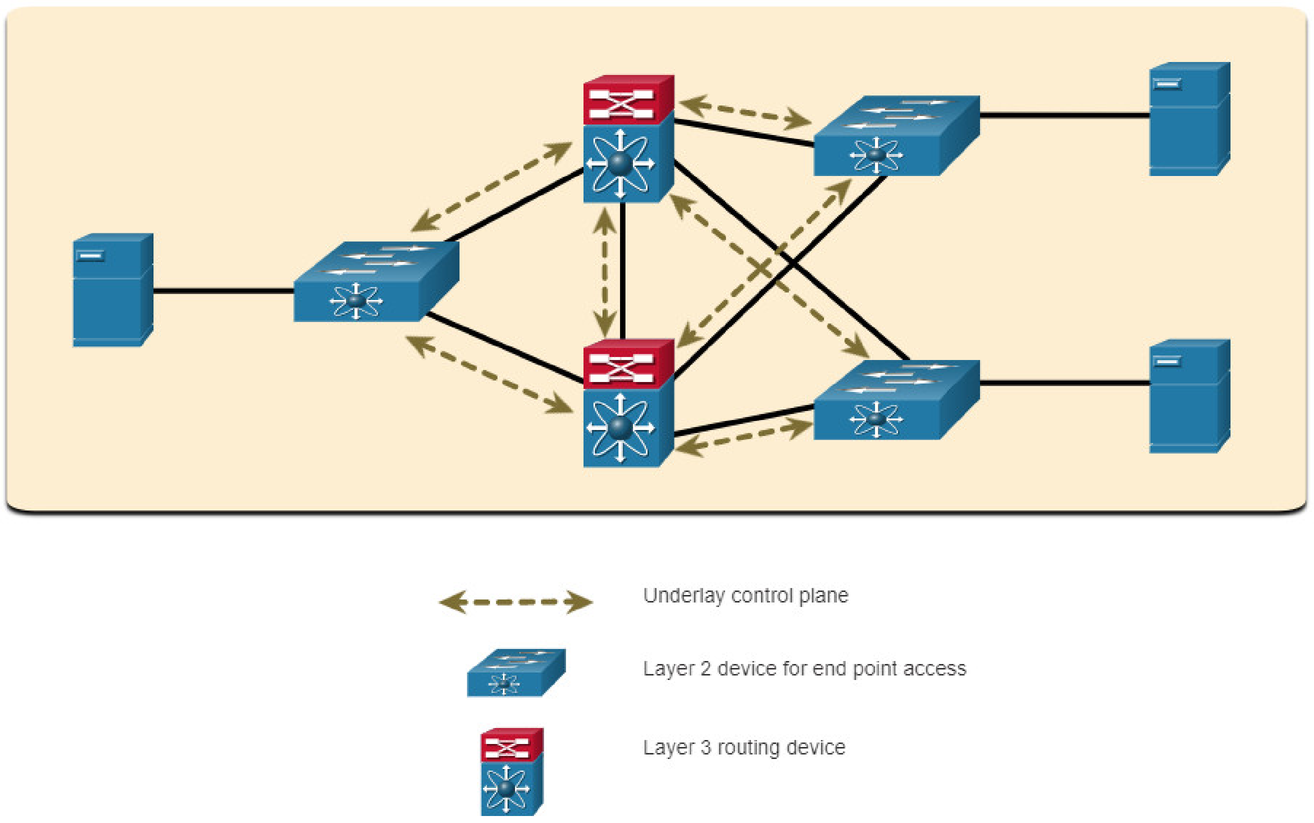
Provisioning of Wireless Access Points (CAPWAP) occur. Using an IBN solution, the network administrator can specify through policies exactly what happens in the overlay control plane. Notice that how the switches are physically connected is not a concern of the overlay.

Example Overlay Network



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, as shown in the figure. 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.

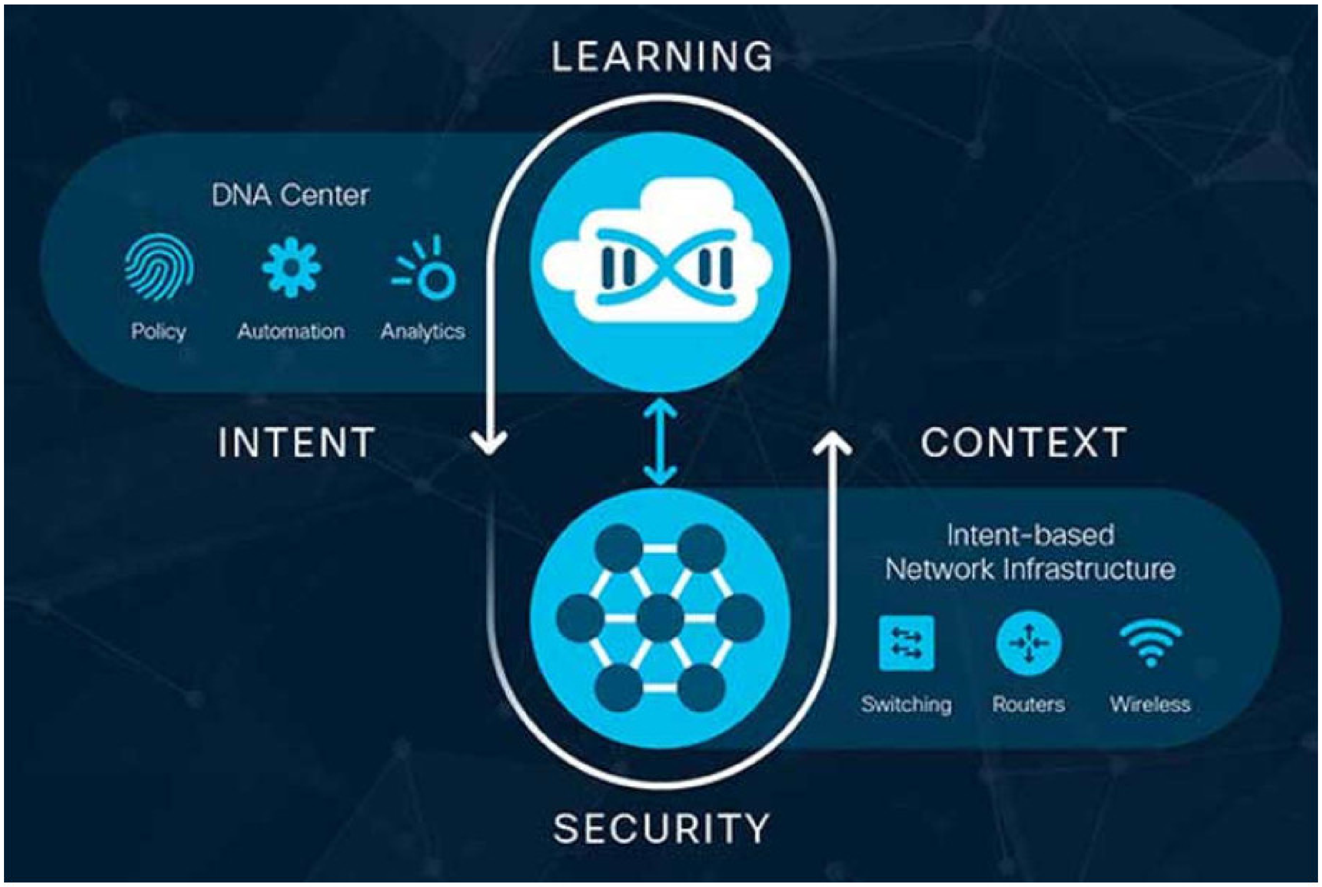
Example Underlay Network



14.6.4. Cisco Digital Network Architecture (DNA)

Cisco implements the IBN fabric using Cisco DNA. As displayed in the figure, 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 DNA Continuous Implementation of Business Intent



Cisco DNA is a system that is constantly learning, adapting to support the business needs. The table lists some Cisco DNA products and solutions.

Cisco DNA Solution	Description	Benefits
SD-Access	• First intent-based enterprise networking solution built using Cisco DNA. • It uses a single network fabric across LAN and WLAN to create a consistent, highly secure user experience. • 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.

Cisco DNA Solution	Description	Benefits
SD-WAN	• It uses a secure cloud-delivered architecture to centrally manage WAN connections. • 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. • Achieve greater agility and cost savings through easier deployments and transport independence.
Cisco DNA Assurance	• Used to troubleshoot and increase IT productivity. • It applies advanced analytics and machine learning to improve performance and issue resolution, and predict to assure network performance. • It provides real-time notification for network conditions that require attention.	• Allows you to identify root causes and provides suggested remediation for faster troubleshooting. • The Cisco DNA Center provides an easy-to-use single dashboard with insights and drill-down capabilities. • 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. • 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. • Gain 360-degree visibility through real-time analytics for deep intelligence across the network. • Lower complexity with end-to-end security.

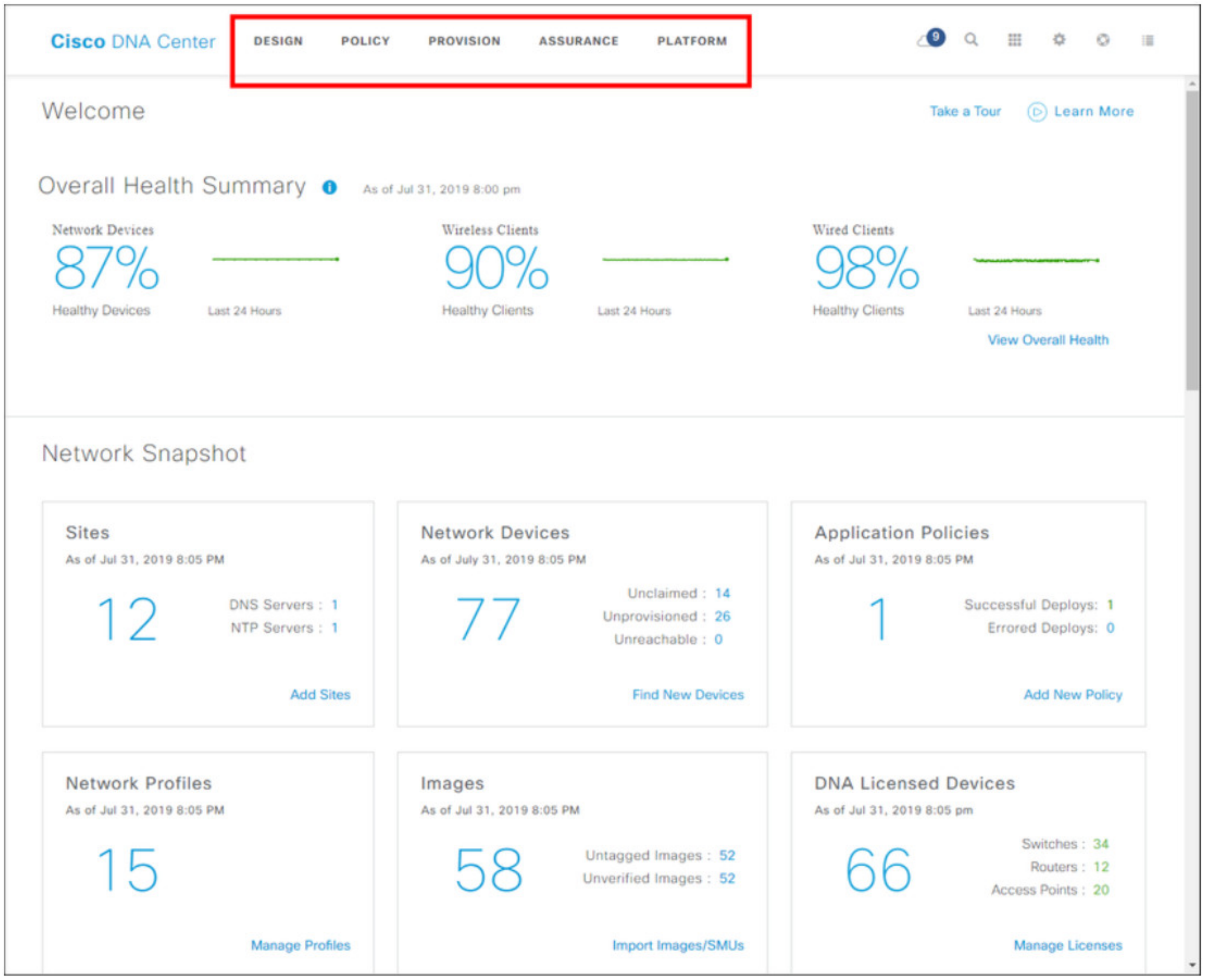
These solutions are not mutually exclusive. For example, all four solutions could be deployed by an organization.

Many of these solutions are implemented using the Cisco DNA Center which provides a software dashboard for managing an enterprise network.

14.6.5. 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, as shown in the figure. From here, the network administrator can quickly drill down into areas of interest.



Welcome

Overall Health Summary As of Jul 31, 2019 8:00 pm

Network Devices 87% Healthy Devices Last 24 Hours

Wireless Clients 90% Healthy Clients Last 24 Hours

Wired Clients 98% Healthy Clients Last 24 Hours

Network Snapshot

Sites As of Jul 31, 2019 8:05 PM 12 DNS Servers : 1 NTP Servers : 1

Network Devices As of July 31, 2019 8:05 PM 77 Unclaimed : 14 Unprovisioned : 26 Unreachable : 0

Application Policies As of Jul 31, 2019 8:05 PM 1 Successful Deploys: 1 Errored Deploys: 0

Network Profiles As of Jul 31, 2019 8:05 PM 15

Images As of Jul 31, 2019 8:05 PM 58 Untagged Images : 52 Unverified Images : 52

DNA Licensed Devices As of Jul 31, 2019 8:05 pm 66 Switches : 34 Routers : 12 Access Points : 20

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

14.6.6 Video – DNA Center Overview and Platform APIs

This is Part One of a four-part series demonstrating the Cisco DNA Center.

Part One is an overview of the Cisco DNA Center GUI. It includes design, policy, provision, and assurance tools used to control multiple sites and multiple devices.

Click Play in the figure to view the video.

14.6.7 Video – DNA Center Design and Provision

This is Part Two of a four-part series demonstrating the Cisco DNA Center.

Part Two is an overview of the Cisco DNA Center design and provision areas.

Click Play in the figure to view the video.

14.6.8 Video – DNA Center Policy and Assurance

This is Part Three of a four-part series demonstrating the Cisco DNA Center.

Part Three explains the Cisco DNA Center policy and assurance areas.

Click Play in the figure to view the video.

14.6.9 Video – DNA Center Troubleshooting User Connectivity

This is Part Four of a four-part series demonstrating the Cisco DNA Center.

Part Four explains how to use Cisco DNA Center to troubleshoot devices.

Click Play in the figure to view the video.

14.7. Module Practice and Quiz

14.7.1. What did I learn in this module?

Automation is any process that is self-driven, reducing and potentially eliminating, the need for human intervention. Whenever a course of action is taken by a device based on an outside piece of information, then that device is a smart device. For smart devices to “think”, they need to be programmed using network automation tools.

Data formats are simply a way to store and interchange data in a structured format. One such format is called Hypertext Markup Language (HTML). Common data formats that are used in many applications including network automation and programmability are JavaScript Object Notation (JSON), eXtensible Markup Language (XML), and YAML Ain’t Markup Language (YAML). Data formats have rules and structure similar to what we have with programming and written languages.

An API is a set of rules describing how one application can interact with another, and the instructions to allow the interaction to occur. Open/Public APIs are, as the name suggests, publicly available. Internal/Private APIs are used only within an organization. Partner APIs are used between a company and its business partners. 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), and JavaScript Object Notation-Remote Procedure Call (JSON-RPC).

A REST API 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”. RESTful APIs use common HTTP methods including POST, GET, PUT, PATCH and DELETE. These methods correspond to RESTful operations: Create, Read, Update, and Delete (or CRUD). Web resources and web services such as RESTful APIs are identified using a URI. A URI has two specializations, Uniform Resource Name (URN) and Uniform Resource Locator (URL). 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. The different parts of the API request are API server, Resources, and Query. Queries can include format, key, and parameters.

There are now new and different methods for network operators 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 use RESTful API requests to automate tasks and scale across thousands of devices. Characteristics of the network that benefit from automation include software and version control, device attributes such as names, addressing, and security, protocol configurations, and ACL configurations. Configuration management tools typically include automation and orchestration. Orchestration is the arranging of the automated tasks that results in a coordinate process or workflow. Ansible, Chef, Puppet, and SaltStack all come with API documentation for configuring RESTful API requests.

IBN builds on SDN, taking a software-centric, fully automated approach to designing and operating networks. Cisco views IBN as having three essential functions: translation, activation, and assurance. The physical and virtual network infrastructure is a fabric. The term fabric describes an overlay that represents the logical topology used to virtually connect to devices. The underlay network is the physical topology that includes all hardware required to meet business objectives. 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. Cisco DNA Center is the foundational controller and analytics platform at the heart of Cisco DNA. Cisco DNA Center is a network management and command center for provisioning and configuring network devices. It is a single interface hardware and software platform that focuses on assurance, analytics, and automation.

14.7.2 Module Quiz – Network Automation

Download Slide Powerpoint (PPT)

CCNA 3 v7.0 Curriculum: Module 14 - Network Automation.pptx

1 file(s) 1.75 MB

Download

Tags:ccna 3 v7 modules