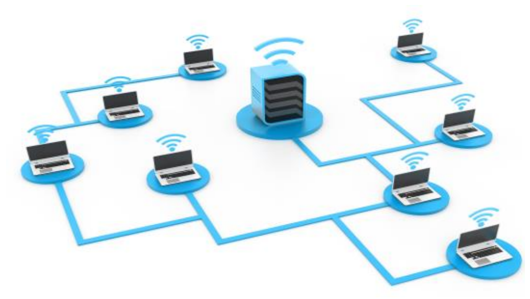
Automation and Programmability



Exam Topics

- Traditional Networking
- Data Plane
- Control Plane
- Management Plane
- Software-Defined Networking
- Controller-Based Architecture
- Physical Underlay
- Layer 2 Overlay
- Layer 3 Overlay
- Introduction to Automation
- Puppet, Chef and Ansible Tools
- REST API Architecture
- Stack Communication
- HTTP Verbs
- JSON Encoding
- Cisco DNA Center

Traditional Network Architecture

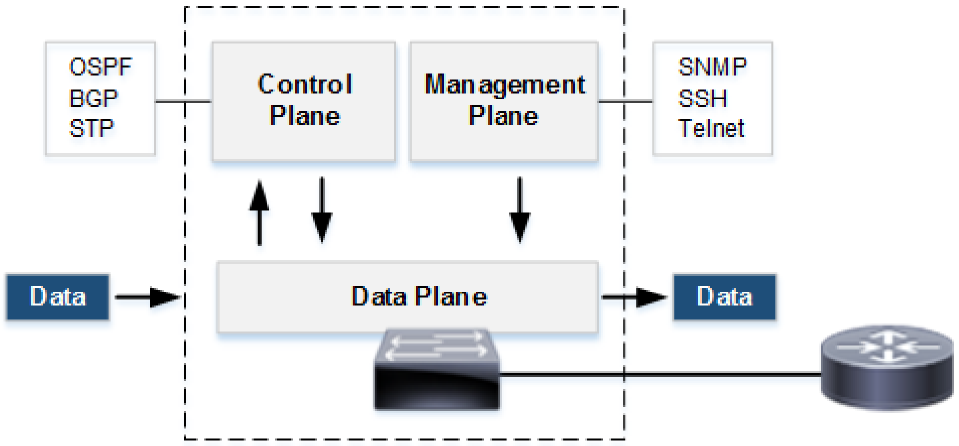
The purpose of any network is to enable data communication between host endpoints. That is accomplished with infrastructure devices that communicate via network protocols. The network operational model can be described using the concept of planes. Each network device within a traditional network has a data plane, control plane and management plane. It is a functional model that describes the dynamics of data communications and networking services. There are also differences between traditional and newer controller-based architecture, that is evident with the location of operational planes.

Data Plane

The data plane is only responsible for forwarding of endpoint data traffic between network interfaces. All data plane traffic is **in-transit** between neighbors, and not associated with communication protocols. It is not handled by the processor as a result. For example, routing tables created by the control plane are used by the data plane to select a route. The packet is then forwarded to a next hop neighbor address.

- MAC learning and aging
- MAC address table lookup
- Routing table lookup
- ARP table lookup
- MAC frame rewrite

Figure 1 Traditional Network Architecture



Similarly, MAC address and ARP tables created by the control plane, are used by the data plane to forward traffic. While all three planes exist on all network devices, the services provided are based on the device class. For example, only routers and L3 switches support routing tables, ARP tables and frame rewrite. Conversely, all switches create MAC address tables while routers do not.

Control Plane

The control plane is responsible for building network tables used by the data plane to make forwarding decisions. Control plane protocols only communicate with directly connected neighbors. It is only the processor that handles inbound and outbound control plane traffic. There are routing protocols that build routing tables from neighbor advertised routes for Layer 3 connectivity. Some common examples of Layer 3 control plane protocols include OSPF, EIGRP, BGP, and ICMP.

- Network tables
- Path selection
- Frame switching
- Link negotiation
- Error messages

Control plane protocols also enable interconnection of switches within Layer 2 domains. For example, STP enables a loop free topology between multiple switches. There is dynamic trunk negotiation between neighbor switches and EtherChannels. Examples of Layer 2 control plane protocols include STP, DTP, LACP, and CDP. Network switches create MAC address tables for frame switching within Layer 2 domains.

Management Plane

The management plane is responsible for configuration and monitoring of network devices. There are various application protocols that are used to manage the network. For example, SSH is initiated to the management plane of a router to configure network interfaces. SNMP sends traps to a network management station to alert on operational status of interfaces.

- Configuration
- Monitoring
- Automation
- Programmability

There are newer protocols such as NETCONF that enable automation of management functions. Similar to the control plane, is the fact that all management plane protocols must be handled by the processor. Some other examples of management plane protocols include TFTP, Telnet, RESTCONF, Syslog, NTP, DNS, and DHCP.

1. The management plane initiates a session with the local router to configure OSPF and enable network interfaces.

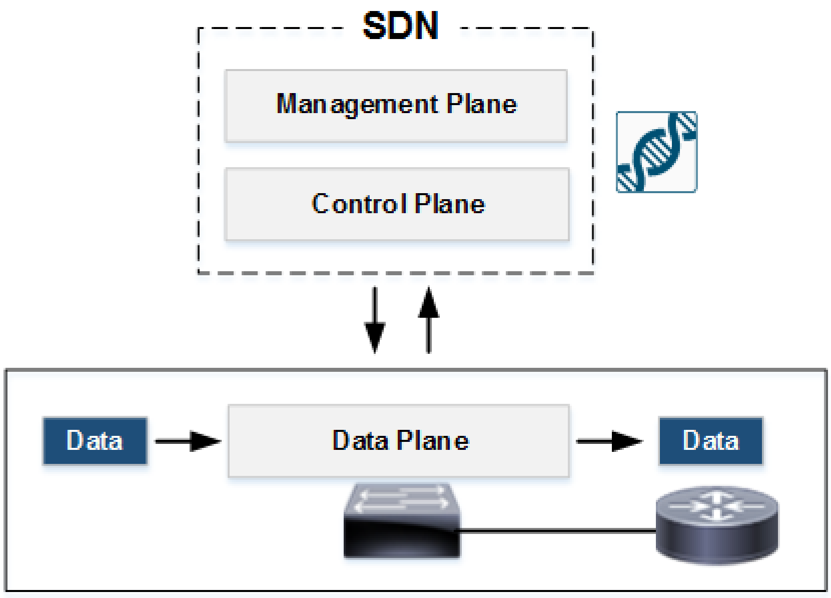
2. The control plane has a routing table with a route that includes a next hop address and local exit interface.

3. The data plane does a routing table lookup for the next hop address associated with a destination subnet. The data plane then forwards all packets to neighbor with next hop address.

Software-Defined Networking (SDN)

Software Defined Networking (SDN) is an architecture that separates the control plane from the data plane. Cisco IOS software is moved to an SDN controller. That decouples the control plane from hardware and enables direct programmability of all network devices. The controller communicates via agents installed on devices. The same functions are provided as with traditional networking architecture for each operational plane. Figure 6-2 illustrates how the management plane is also moved to the controller as well.

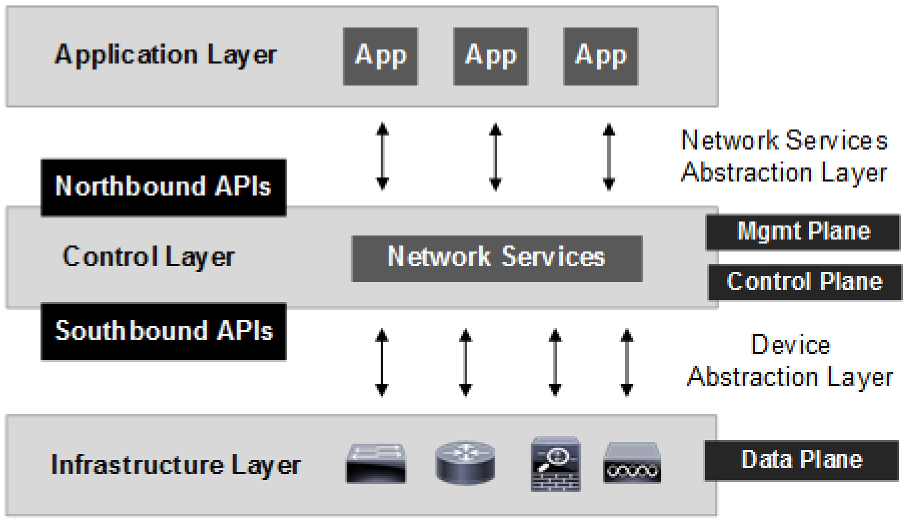
Figure 2 SDN Operational Planes



It is similar to a hypervisor layer that abstracts (separates) server hardware from application software. There is a centralized control plane that is software-based with an underlying physical data plane transport. SDN enables overlays and programmable devices for management with centralized policy engine and global view.

- SDN decouples the control and data plane.
- Control plane is software-based and not a hardware module.
- SDN controller is a centralized control plane with a policy engine.
- Network infrastructure is an underlay for programmable fabric.

Figure 3 SDN Architecture Layers



SDN Components

The SDN controller provides centralized management where the network appears as a single logical switch. Network services are dynamically configurable when the control plane is moved from physical infrastructure to a software-based SDN controller with API modules. The northbound and southbound APIs enables communication between applications and network devices.

Table 1 SDN Components

Attribute	Description
SDN Application	software that send requests to SDN controller via northbound APIs.
Northbound API	software on controller that expose SDN applications to controller services.
Controller	centralized control software that translates requests from SDN applications to network devices.
Southbound API	software that sends messages to communicate with agents on network devices for programmability.
Infrastructure	physical and virtual programmable network devices that provide data plane (forwarding) services.

SDN controllers communicate with physical and virtual network devices via southbound APIs. Conversely, communication from controller to SDN applications is via northbound APIs. There is a policy engine configured on a controller for orchestration and automation of network services.

- **Programmability** - network is directly programmable because it is decoupled from infrastructure and data plane forwarding.
- **Agility** - abstracting control plane from data plane enables dynamic configuration to modify traffic flows as network conditions change.
- **Centralized Management** - network intelligence is centralized in software-based SDN controllers. The global network appears to applications and policy engines as a single logical switch.
- **Automation** - dynamic configuration (provisioning) of network devices and software upgrades is based on APIs.

Network Functions Virtualization (NFV) increase agility by decoupling network services from proprietary hardware and moving it to software modules on SDN controllers. That makes it easier to provision, automate, and orchestrate network services such as DNS, firewall inspection and network address translation.

Advantages of Programmability

- Workload mobility
- Elastic auto scaling
- Dynamic policy provisioning
- Security-based diversion
- Dynamic path selection
- Services insertion
- Automated push configuration
- Dynamic bandwidth allocation
- Wireless RF optimization
- Network security updates

The advantage of programmability include automation and rapid deployment of new services and applications. Turn up of a new branch office or an application is now accomplished in minutes. Newer Cisco devices support programmable ASICs. Open APIs translate between application and hardware to initialize, manage and change network behavior dynamically.

New requirements now include on-demand bandwidth, dynamic security and elastic capacity. In addition rapid cost effective deployment of applications and services. The provisioning of wired and wireless services requires automated turn-up of network services, push configuration, automatic monitoring and real-time analysis.

Network Overlays

Cisco has recently developed SD-Access fabric architecture for data center and enterprise connectivity. The purpose is to enable automation, programmability and mobility for physical and virtual platforms. It is comprised of an underlay, fabric overlays and Cisco DNA Center.

Fabric Underlay

The fabric is comprised of a physical underlay designed for high-speed transport of traffic. It is characterized by network devices, topology and protocols for communication. There is a common underlay that provides transport for overlay traffic. That would include control plane protocols such as STP, DTP, OSPF, EIGRP and ARP.

- Network infrastructure used for transport of all data traffic
- Comprised of network devices, protocols and configuration
- Network devices must support programmability with agents
- Physical underlay operation is independent of overlays

Fabric Overlay

There is also path virtualization enabled with fabric overlays that are built on top of (or over) the underlay. Overlays create a virtual topology across a physical underlay infrastructure with encapsulation techniques that create tunnels. That essentially enables route and address isolation, that is independent of underlay and other overlays. Encapsulation is nothing more than adding outer header/s to original payload that is not visible to network devices when in-transit.

- Virtual topology with interconnects between nodes
- Encapsulation (tunnel) creates path virtualization
- Network address overlap and route isolation enabled
- Overlays are operationally independent of underlays

Consider that overlays logically create single point-to-point connections. That same topology has multiple physical connections between switches. The purpose of overlays are to solve limitations inherent with physical switching domains such as STP, routing loops, broadcasts and address overlap. They also enable multi-tenant service, enhanced mobility, seamless connectivity and automation.

Table 2 Underlay vs Overlay

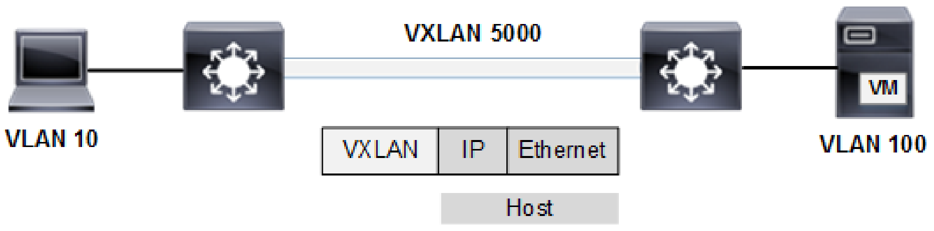
Underlay	Overlay
physical	virtual
single-tenant	multi-tenant
unique address space	overlap address space
native transport	tunneling

Layer 2 Overlay

Within the fabric architecture there is support for Layer 2 and Layer 3 overlays. Layer 2 overlays are designed to emulate a physical topology for the purpose of extending Layer 2 domains. For example, connecting two servers on different switches that are assigned to the same VLAN. The solution is a VXLAN overlay to enable a virtual connection between servers. It is common to have web-based applications with multiple servers that are often in different locations.

- Emulates a physical switching topology with virtual overlay
- Extend Layer 2 domains between switches and locations
- Enable address isolation and overlapping between domains
- Tunnels terminate at leaf switches for campus deployment

Figure 4 VXLAN Fabric Overlay



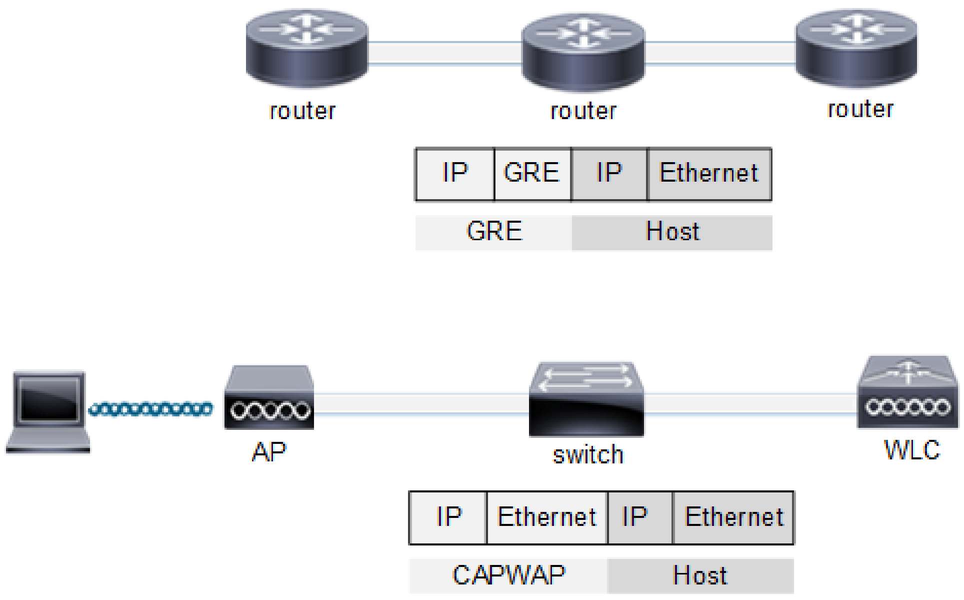
VXLAN is a data plane overlay that encapsulates host packets for communication across fabric. As an overlay, it requires the transport services of a physical underlay infrastructure. In our example, the tunnels are terminated at fabric edge switches. There is a common underlay for data plane forwarding, however the underlay topology is independent of overlay topologies. As a result, underlay and overlay maintain separate data and control planes.

Layer 3 Overlay

Layer 3 overlays enable data plane forwarding across a fabric between different subnets. There is the advantage as well of isolation from the underlay limitations associated with MAC flooding and spanning tree protocol loops. Tunnels are created with encapsulation of host packets. Some examples include VPN, MPLS, GRE, CAPWAP and VRF.

- Routing-based overlay for IP connectivity across fabric
- Isolates broadcast domains to each network device
- IP tunnel terminates at host endpoint or network device
- Logical point-to-point topology between tunnel endpoints

Figure 5 GRE and CAPWAP Overlays



Automation Fundamentals

The advent of network programmability and automation tools is radically changing how network infrastructure is managed. Compared with traditional networking, automation has astonishing advantages for physical and virtualized network services. Network automation lowers operational costs, enables deployment agility, and unified policies.

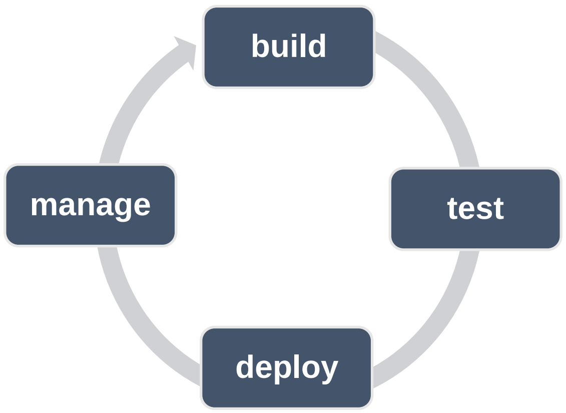
Advantages of Network Automation

- Minimize network outages
- Enable deployment agility
- Lower operational costs
- Unified security policies
- Software compliance

The most common cause of network downtime is user error. There are significantly less errors with configuration changes and deployment of network infrastructure. There is a globally centralized view of the network that is fundamental to SDN architecture.

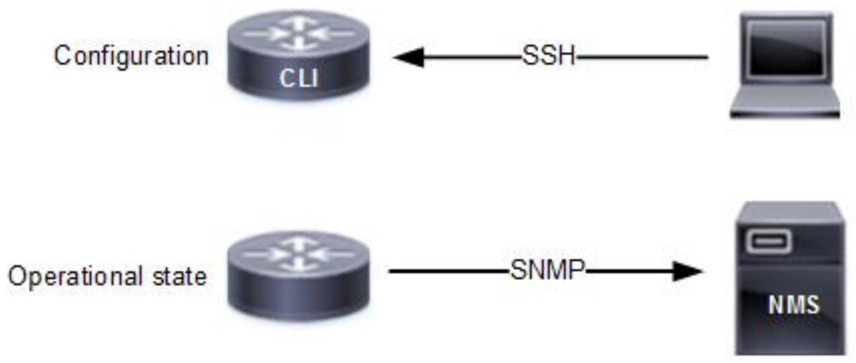
Network administrators can push standard configurations out to new network devices and update configuration on existing infrastructure. The audit of device configuration or software versions for compliance before update is much faster with automation.

Figure 6 Automation Life Cycle



Configuration management tools such as Puppet, Chef and Ansible are used to enable automation for on-premises and cloud-based services. They were developed originally for managing cloud computing and network virtualization infrastructure. Figure 6-6 illustrates how automation is based on quality assurance. There is a build, test and deploy model that minimizes errors and downtime before configuration changes are pushed to devices.

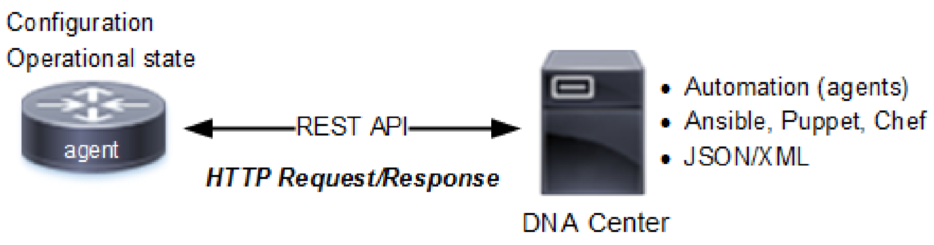
Figure 7 Traditional Networking Management



Previously, traditional networking was based on a silo view where each network device was statically managed separately. Having a centralized, real-time network view is fundamental to automation. There is much more accomplished, in less time and lower cost with minimal network outages.

The new architecture is based on agents that are deployed to network devices. They enable communication with a controller such as Cisco DNA Center via REST APIs. Multiple different automation tools are available such as NETCONF, RESTCONF, Ansible, Puppet and Chef. They are southbound APIs within SDN framework that provide features to automate configuration, software updates and other services.

Figure 8 Controller-Based Management Automation



Common Automation Management Tasks

- Unified infrastructure, software and security policies
- Push new configurations for physical/virtual infrastructure
- Standardize and update existing network device configuration
- Enhanced software compliance and version control
- Maintenance backup of system images and scripts
- Network discovery, detection and analysis
- Programmable scripting instead of manual CLI

Table 3 Traditional vs Automation

Traditional	**Automation**
CLI	scripting
manual	dynamic
slower	faster
error-prone	assurance
deploy	test and deploy
not scalable	scalable
physical server	physical + virtual machine

Configuration Management Tools

There are various open source configuration management tools that enable automation and orchestration of basic and complex tasks. They were originally developed for cloud computing however they are used to manage on-premises infrastructure as well. Some of the most popular automation tools include Puppet, Chef and Ansible.

Each of them have advantages, disadvantages and specific architecture. Configuration management for network infrastructure is mostly comprised of provisioning, compliance and maintenance tasks. Consider that traditional networking is all based on manual CLI access and SNMP operational monitoring of physical devices.

With the advent of virtualization, network administrators are also managing network servers that are deployed as virtual machines (VM). That could include virtual appliances in the cloud or network services at the data center such as TFTP server and wireless LAN controller.

Example Automation Tasks

- Create new VLANs on switches
- Create management interface on switches
- Deploy initial configuration to multiple routers
- Update routers based on new PSIRT security alert
- Backup startup configuration scripts to TFTP server

The primary difference in architecture is how network nodes (client) are updated. Puppet and Chef are based on agents that are installed on clients and communicate with a centralized server (Puppet Master). Ansible has an agentless architecture where no software is installed on client nodes. The agent-based model is called a "pull mode" where clients pull or download a configuration or software update from a centralized server. The "pull mode" of Ansible relies on dynamic polling from a server.

They are open standard tools developed from cloud environments. One of the advantages of each platform is reusable template scripts that have been developed for a variety of common tasks. The protocols from Table 6-4 describe how client and server communicate. Puppet is a full-fledged configuration management tool with monitoring of state consistency. Puppet can also test scripts before making changes in simulation mode. All tools can create automated scripts for provisioning, compliance, and maintenance tasks.

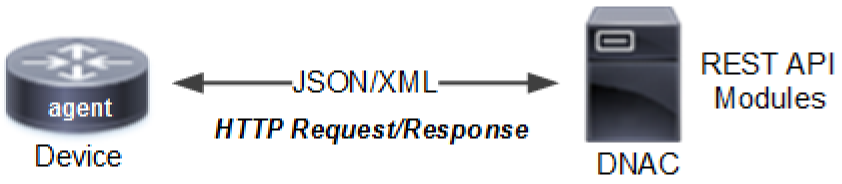
Table 4 Comparing Puppet, Chef, and Ansible

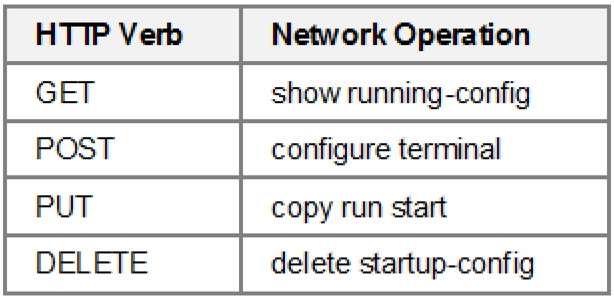
Puppet	**Chef**	**Ansible**
agent	agent	agentless
pull	pull	push
HTTPS	SSH	SSH
Ruby	Ruby	Python
modules	recipes	playbook (YAML)
complex	complex	easy

REST API Architecture

The proliferation of web-based applications and cloud computing has led to development of Representational State Transfer (REST) architecture. It is a framework with rules for creating web-based services. It is based on stateless operations (verbs) that are performed on objects (resources) defined with URLs.

Figure 9 REST API Architecture





HTTP Verb	Network Operation
GET	show running-config
POST	configure terminal
PUT	copy run start
DELETE	delete startup-config

REST API is an example of northbound API that enables communication between SDN controller and applications. HTTP requests and responses are sent between machines based on GET, POST, PUT, DELETE verbs. They are used for inter-application communication. The server has no knowledge (stateless) and does not differentiate between single tasks or an entire application. REST API is an application programming interface compliant with REST architecture.

1. **Define Resources:** URL (web page, image, document, video, email)

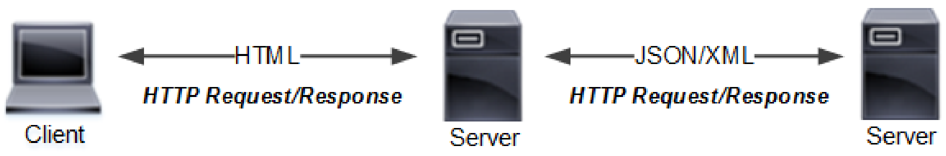
2. **Assign HTTP Verbs:** GET, POST, PUT, DELETE

3. **Select Encoding Type:** HTML, JSON, or XML data encoding format

For example, you could develop a REST API to rent computer labs where customers request and book time online. Customers would start from the top-level navigation web page to request (GET) lab time. The application then responds (POST) with available times and dates.

The customer could book a reservation (POST) time slot for a particular day. The customer then decides to change lab time reserved with a PUT request that updates existing data. There is also DELETE operation to cancel a booking.

Figure 10 RESTFUL API Architecture



HTTP Verb	Operation
POST	Create
GET	Read
PUT	Update
DELETE	Delete

REST API is often comprised of multiple web pages with features that are accessed through HTTP verbs. The data encoding type allow machines to read payload data in a common format. The encoding types for data interchange include JSON, HTML and XML. Communication between client browser and server is via HTML. In addition, server to server or server to device data exchange is encoded with JSON or XML.

API Stack Communication

REST API applications are designed for machine to machine (M2M) communication. The components of an application include HTTP methods (verbs), encoding type, network transport and authorization header. There is no encryption cipher defined with REST API. Instead encryption is provided with HTTPS where desired. The following describes API stack communication with HTTP application layer.

- HTTP/S Verbs = GET/POST/PUT/DELETE
- Authentication Method
- Encoding = JSON
- Transport = TCP 80/443
- PHY = Ethernet/Wireless

HTTP Verbs

Verbs are HTTP operations that define some action to read, add or modify resources on a server. The HTTP operations access resources represented as URLs and return payload response to a client, server or network device. Cisco DNA Center is essentially a server that manages physical and virtual network devices.

GET - This retrieve operation only requests to read a resource from the server such as a web page. The response header from a server or network device includes data content, with no state change.

POST - This operation is used to create new data or provide new data. Any client, server or network device can initiate POST operation and response is returned with updated data state.

PUT - This operation is used to send response to update or replace existing data on a server or network device.

DELETE - This operation removes data from a server or network device.

CRUD was primarily developed to manipulate database records for a variety of traditional application platforms. Recently, it has been adapted as well for web-based applications. CRUD methods are mapped to HTTP verbs for creating REST API and compliance with REST architecture.

Table 5 HTTP Verbs vs CRUD Operations

HTTP Verb	CRUD Operation
POST	CREATE
GET	READ
PUT	UPDATE
DELETE	DELETE

You can define a REST API for creating web services based on CRUD operations as a result. For example, consider an online shopping cart where CRUD is used to READ a web page where some CCNA books are listed. The next operation is CREATE to checkout and send payment for selected item. You could then change your shipping location with an UPDATE operation. The DELETE operation is used to remove/cancel a shopping cart session that was started.

JSON Data Encoding

REST does not specify any encoding type when creating a REST API for web services. In fact, there are various encoding types available that enable data interchange for platform independent data sharing. Each data encoding also specifies character sets that are supported. The purpose of encoding is to define a format for data that enable data interchange between disparate systems.

JSON is a readable text-based encoding method comprised of objects, arrays, and name/value pairs. It is based on a standardized syntax that enables data interchange between disparate systems. The following are some primary rules for correct syntax when creating JSON scripts.

- integer number value with double quotes are not allowed
- null value with double quotes are not allowed
- strings are used to represent values such as phone numbers
- square brackets [ ] create an array list within an object
- curly brackets { } create an object with name/value pairs
- comma-separated list of array values or name/value pairs

The following is an example of a JSON object that has the hostname and IP addressing for a data center router. JSON objects are comprised of single or multiple name/value pairs that are separated by commas. For example, name = hostname and value = router-1.

Example 1 JSON Object

```
{
“DC-Routers”: {
“hostname”: “router-1”,
“ip_address” : “192.168.1.1”,
“subnet_mask” : “255.255.255.0”
}
}
```

JSON array is basically a list of items that are associated with an array name. For example, an array called network interfaces could be created with multiple interfaces on a switch line card. An array can also comprise multiple objects, where each list item is an object separated by commas. There is a single error only with the JSON script. It is a comma after curly bracket of second object }, that is not required. The last object in an array has a matching curly bracket only. The following example illustrates in bold where array starts and ends with square [ ] brackets. It is comprised of a list of network interfaces on a switch.

Example 2 JSON Array

```
{
"hostname": "switch-1",
"interfaces": [
"Loopback0",
"Serial1/0",
"Ethernet1/1",
"Ethernet2/1"
]
}
```

Table 6 Data Encoding Types

JSON	XML	HTML
text-based	text-based	text-based
client-server server-server	server-server	client-server
faster	slower	slower
less secure	more secure	more secure
text and number	multiple data types	multiple data types
key values, arrays	tree structure	tree structure
UTF-8, ASCII	UTF-8, ISO, ASCII	UTF-8, ISO, ASCII
REST API	REST API	REST API

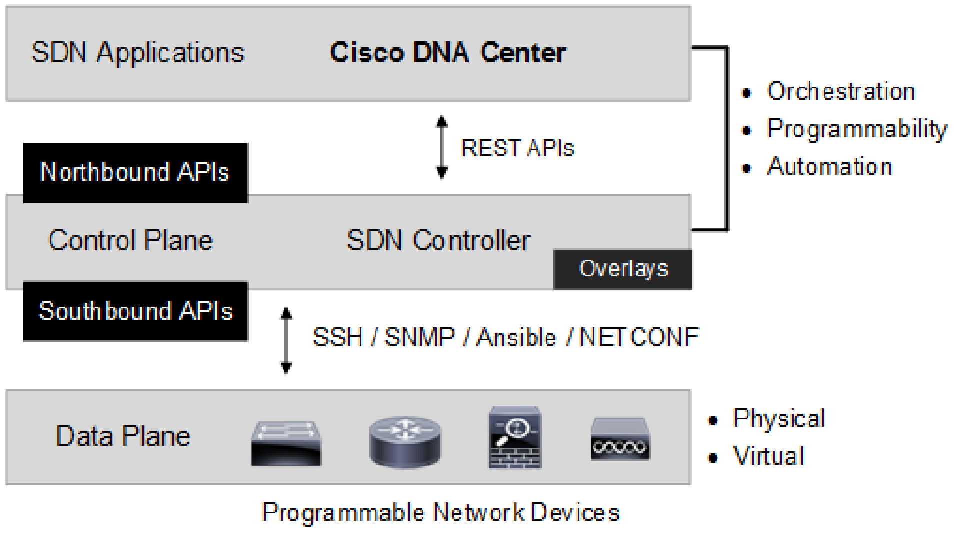
Character Encoding

The original character set standard was ASCII based on a character set with 128 numbers, letters and symbols. It was superseded by UTF-8 with the advent of web-based applications and the need for an expanded character set. In fact, by some estimates, UTF-8 is currently the default character set used for approximately 95% of all web pages. New versions of the same standard now include UTF-16 and UTF-32 character set. The default character set for JSON, XML and HTML is UTF-8.

Cisco DNA Center

Cisco DNA Center is a network management solution that is based on SDN architecture. DNA Center enables orchestration, automation and programmability of wired, wireless, and virtualized services. The level of complexity has increased with virtualization, mobility and on-premises connection to cloud data centers.

Figure 11 Cisco DNA Center Architecture (DNAC)



Cisco DNA Center is a full-fledged management application that includes a Cisco proprietary SDN controller. It provides a global view of all physical and virtual machines (VM) that are centrally managed. The functional components of Cisco DNA Center include design, provision, policy, and assurance. Intent-based REST API modules are enabled for discovery, configuration, automation, and monitoring purposes. That is supported for both fabric and non-fabric infrastructure.

DNA Center is installed on an appliance that is connected to a shared services block at a data center. There is also ISE, DHCP, DNS and NTP server deployed to a shared services block. It is common to also install configuration management tools such as Ansible or NETCONF as well to augment configuration automation tasks.

Table 7 Traditional vs Controller-Based Networking

Traditional	**Controller-Based**
silo	global
distributed	centralized
physical	logical
static	programmable
single-task	orchestration
local	unified
device	network
trial and error	assurance
point-in-time	real-time

Cisco DNA Center enables northbound REST APIs and workflows to manage fabric and non-fabric devices. REST APIs communicate with network devices and perform functions via HTTP methods (GET, POST, PUT, DELETE). For example, to create a new fabric site would require REST API POST request. Any updates to a new site such as changing fabric site name would use PUT method. GET request is used to read device configuration or collect operational state. JSON is the data encoding method supported with Cisco Intent-based REST APIs.

Network discovery is started to populate inventory with all detected network devices. GET method collects information on clients, sites, topology, devices and operational state. It reads resources such as configuration script while POST method creates a new configuration.

Next, policies are defined with configuration settings and security access for each device class and functional roles. The provisioning workflows add network devices to sites, push configuration based on policies, and create fabric domains. Finally there is assurance to enable performance monitoring, operational state and management of all network elements.

CCNA Whiteboard

There is a 15-minute tutorial before the CCNA exam starts. You have that time to create your whiteboard notes on laminated paper provided at the testing center. Learn how to create your own customized whiteboard with **subnetting and binary conversion tables**. CCNA exam whiteboard provides an essential quick study review to do your best. The whiteboard should include subnetting table, binary conversion table, IOS commands, port numbers, wireless, SDN, route selection, AD values, OSPFv2, ACLs, Cisco default settings and anything else you often forget.