PassQueen
a.k.a dumpsqueen.com

Automating Cisco Enterprise Solutions (ENAUTO)

Cisco 300-435

Version Demo

Total Demo Questions: 20

Total Premium Questions: 201

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Network Programmability Foundation	24
Topic 2, APIs and Protocols	70
Topic 3, Cisco Platforms and Application Development	88
Topic 4, Infrastructure Automation	19
Total	201

QUESTION NO: 1

```
# Simple Application to run a few commands on a Cisco Device
ipaddresses = ['192.168.0.1', "192.168.0.5", "10.10.10.10"]
username = "admin"
password = "cisco123"
commands_to_run=["show ver", "show ip interface brief"]
Debug = True


for device in ipaddresses:
      print ("Logging into "+device+", using "+username+"/"+password)

      # We want to execute commands on our device only if Debug=True

      for commands in commands_to_run:
            print ("    Executing "+commands+" on device: "+device)
```

Refer to the exhibit. What is the expected output from the Python code?

A.
```
Logging into 192.168.0.1, using admin/cisco123
Logging into 192.168.0.5, using admin/cisco123
Logging into 10.10.10.10, using admin/cisco123
      Executing show ver on device: 192.168.0.1
      Executing show ip interface brief on device: 192.168.0.1
      Executing show ver on device: 192.168.0.5
      Executing show ip interface brief on device: 192.168.0.5
      Executing show ver on device: 10.10.10.10
      Executing show ip interface brief on device: 10.10.10.10
```

B.
```
Logging into 192.168.0.1, using admin/cisco123
Logging into 192.168.0.5, using admin/cisco123
Logging into 10.10.10.10, using admin/cisco123
```

C.
```
Simple Application to run a few commands on a Cisco Device
Loggig into 192.168.0.1, using admin/cisco123
We want to execute commands on our device only if Debug=True
      Executing show ver on device: 192.168.0.1
      Executing show ip interface brief on device: 192.168.0.1
Logging into 192.168.0.5, using admin/cisco123
We want to execute commands on our device only if Debug=True
      Executing show ver on device: 192.168.0.5
      Executing show ip interface brief on device: 192.168.0.5
Logging into 10.10.10.10, using admin/cisco123
We want to execute commands on our device only if Debug=True
      Executing show ver on device: 10.10.10.10
      Executing show ip interface brief on device: 10.10.10.10
```

D.
```
Logging into 192.168.0.1, using admin/cisco123
      Executing show ver on device: 192.168.0.1
      Executing show ip interface brief on device: 192.168.0.1
Logging into 192.168.0.5, using admin/cisco123
      Executing show ver on device: 192.168.0.5
      Executing show ip interface brief on device: 192.168.0.5
Logging into 10.10.10.10, using admin/cisco123
      Executing show ver on device: 10.10.10.10
      Executing show ip interface brief on device: 10.10.10.10
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: D

Explanation:

The expected output is represented by *Option D*. The provided question payload does not include the actual Python source code or the visible contents of either exhibit; it contains only image placeholders and generic option labels. Therefore, the result cannot be independently recalculated from the text available here. The selection retains the supplied answer key, which identifies this output choice as correct.

In general, determining Python output requires evaluating expressions in execution order, applying Python's data-type and operator rules, and accounting for such details as mutation, iteration order where applicable, string formatting, and print behavior. Python's language reference defines the execution model and expression semantics that govern such results, while the built-in-functions reference documents the behavior of output functions such as `print()`. See the Python expression reference and the Python print() documentation. With the exhibit code and output images available, the expected output should be verified directly against those rules.

QUESTION NO: 2

Which two actions should a script perform when it receives an HTTP 429 response from the Meraki Dashboard API? (Choose two.)

A. Read the Retry-After response header.

B. Retry immediately until the request succeeds.

C. Use retry logic with request pacing or backoff.

D. Replace the API key on every retry.

E. Change the request method from POST to GET.

ANSWER: A C

Explanation:

An HTTP 429 response indicates that the client has sent too many requests within the allowed rate limit. A script should read the `Retry-After` response header when it is present and wait for the indicated interval before retrying the request. This allows the Dashboard API service to control when the client resumes requests.

The script should also implement retry logic with pacing or backoff. Rather than immediately resending many requests, the application should queue work, reduce its request rate, and retry the failed request after the required delay. This is particularly important for bulk workflows, such as configuring switch ports across multiple networks. Cisco documents these practices in the Meraki Dashboard API rate-limit documentation.

QUESTION NO: 3 - (DRAG DROP)

```
GET: https://dnacsrv/api/v1/network-device
{
  "response": [
      {
        "type": "Cisco Catalyst 9300 switch",
        "errorCode": null,
        "family": "Switches and Hubs",
        "location": DC1,
        "role": "ACCESS",
        "macAddress: "a1:2b:30:40:41:50",
        "hostname": "cat_9k_1",
        "serialNumber": "FCW2136L0AK",
        "softwareVersion": "16.6.1",
        "locationName": null,
        "upTime": "13 days, 18:30:33.81",
        "softwareType": "IOS-XE",
        "collectionStatus": "Managed",
        "managementIpAddress": "10.10.22.66",
        "platformId": "C9300-24UX",
        "reachabilityStatus": "Reachable",
        "series": "Cisco Catalyst 9300 Series Switches",
        "snmpContact": "",
        "snmpLocation": "",
```

Refer to the exhibit. A **GET** request is issued to the Cisco DNA Center REST API. Drag and drop the **GET** request URL subpaths from the left onto the objectives on the right. Not all options are used.

ANSWER:

Explanation:

The Cisco DNA Center network-device REST resource supports filtering the device inventory by device attributes returned in the response payload. The response in the exhibit identifies the relevant attributes clearly: `softwareType` and `softwareVersion` describe the operating-system platform and release, `family` identifies the device family grouping, `snmpLocation` holds the location learned or configured through SNMP, and `managementIpAddress` identifies the device management address. The URL query must use the field that represents the exact requested inventory criterion.

To display IOS XE devices at release 16.4.2, both filters are included in the same request and joined with `&`: `softwareType=IOS-XE&softwareVersion=16.4.2`. To locate devices configured through SNMP for DC2, the applicable inventory property is `snmpLocation`, so the filter is `snmpLocation=DC2`. Device-type listing is obtained through the family attribute, using `family=Switches and Hubs`. Finally, a search for the stated device IP must filter on the management address property exposed by the endpoint, using `managementIpAddress=10.222.10.35`.

This approach follows the general Cisco Catalyst Center API model of querying resource collections through URL query parameters. Cisco's current API documentation and interactive resources are available from the Cisco Catalyst Center Developer Center.

QUESTION NO: 4

What is a capability of MV sense MQTT API?

A. Request and subscribe to historical, current, or real-time data

B. Automate the configuration of networking device.

C. Monitor the network and auto adjust for optimal performance

D. Create email alerts for users that violate the security configuration

ANSWER: A

Explanation:

Request and subscribe to historical, current, or real-time data is a capability of the Meraki MV Sense MQTT API. MV Sense exposes camera-generated analytics and event information to external applications through MQTT, a lightweight publish/subscribe messaging protocol. An application can connect to the Meraki MQTT broker, request available data, obtain the current state of supported analytics, and subscribe to topics so that it receives new camera data as it is published. This

model is particularly useful for integrations such as occupancy dashboards, queue-management systems, physical-security workflows, and building-automation applications, because the application can react to analytics events without continuously polling a conventional REST endpoint. The API's data-access model therefore encompasses retrieving prior information where supported, obtaining current information, and receiving ongoing real-time updates through subscriptions. Cisco's Meraki developer resources describe MV camera analytics integration capabilities and the available API ecosystem. See the Cisco Meraki MV Sense documentation and the Cisco Meraki Dashboard API documentation for API and integration guidance.

QUESTION NO: 5

```
module: Cisco-IOS-XE-interfaces-oper
  +--ro interfaces
     +--ro interface* [name]
        +--ro name                          string
        +--ro interface-type?               interfaces-ios-xe-oper:ietf-intf-type
        +--ro admin-status?                 interfaces-ios-xe-oper:intf-state
        +--ro oper-status?                  interfaces-ios-xe-oper:oper-state
        +--ro last-change?                  yang:date-and-time
        +--ro if-index?                     int32
        +--ro phys-address?                 yang:mac-address
        +--ro higher-layer-if*              string
        +--ro lower-layer-if*               string
        +--ro speed?                        uint64
        +--ro statistics
        |  +--ro discontinuity-time?    yang:date-and-time
        |  +--ro in-octets?             uint64
        |  +--ro in-unicast-pkts?       uint64
```

Refer to the exhibit. What is a characteristic of the tree?

A. three optional metrics

B. two leaf-lists

C. ten leaf-lists

D. three containers

ANSWER: A

Explanation:

The tree has three optional `metric` leaves, so **three optional metrics** is the correct characteristic. YANG tree diagrams use concise symbols to communicate the schema structure. A leaf is represented by its name followed by its data type, and a question-mark suffix identifies a node that is optional rather than mandatory. Thus, each displayed `metric?` entry represents an independently optional leaf named `metric`. The fact that the same leaf name appears in separate portions of the hierarchy does not make it a leaf-list; each occurrence belongs to its own parent data node and is evaluated in that local schema context. This is useful in network models because a metric can be supplied where that particular routing, interface, or protocol structure supports it, without requiring the value everywhere in the configuration or state tree. Cisco automation tools that consume YANG models, including RESTCONF and NETCONF clients, use these schema definitions to determine which elements may be omitted from a payload. The YANG language specification defines optional data nodes and the conventions used in tree-format representations in RFC 7950, section 6.2. The standardized tree-diagram notation is also described in RFC 8340.

QUESTION NO: 6

Refer to the exhibits.

```
from device_info import ios_xe1
from ncclient import manager
import xmltodict

netconf_filter = open('filter-ietf-interfaces.xml").read()

if __name__ == '__main__':
   with manager.connect(host=ios_xe1["address"],
                        port=ios+xe1["port"],
                        username=ios+xe1["username"],
                        password=ios+xe1["password"],
                        hostkey_verify=False) as m:

     netconf_reply = m.get(netcong_filter)

     intf_details = xmltodict.parse(netconf_reply.xml)["rpc-reply"]["data"]
     intf_config = intf_details["interfaces"]["interface"]
     intf_info = intf_details["interfaces-state"]["interface"]

     print("")
     print("Interface Details:")
     print(" Name: {}".format(              ["name"]))
     print(" Description: {}".format(intf_config["description"]))
     print(" Type: {}".format(intf_config["type"]["#text"]))
     print(" MAC Address: {}".format(intf_info["phys-address"]))
     print(" Packet Input: {}".format(intf_info["statistics"]["in-unicast-pkts"]))
     print(" Packet Output: {}".format(intf_info["statistics"]["out-unicast-pkts"]))
```

```
<filter>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>GigabitEthernet2</name>
    </interface>
  </interfaces>
  <interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>GigabitEthernet2</name>
    </interface>
  </interfaces-state>
</filter>
```

An engineer creates a Python scripts using ncclient to display interface information. The code must be completed so that it can be tested. Which expression completes the highlighted section in the format call?

A. intf_info

B. intf_config

C. intf_get

D. intf_config[0]

ANSWER: B

Explanation:

`intf_config` is the correct expression because it supplies the interface-specific XML configuration or filter fragment to the string template used by the script. Python's `str.format()` method replaces a placeholder in an XML template with the value passed to it, producing the completed XML payload that ncclient can send in its NETCONF request. In this workflow, the interface-information template provides the surrounding XML structure, while `intf_config` represents the content

inserted at the highlighted placeholder. The resulting XML identifies the relevant interface data in the structure expected by the device's NETCONF/YANG implementation. ncclient's manager operations accept XML filters and payloads as strings, making construction of a valid XML request before the RPC is issued an important step. This lets the script retrieve and display the intended interface information during testing. Cisco's NETCONF programmability material describes using XML/YANG data with NETCONF operations, and the ncclient documentation describes the Python manager interface used to invoke those operations. See Cisco DevNet NETCONF learning material and ncclient Manager API documentation.

QUESTION NO: 7

```
1 {
2   'data':
3     [
4       {
5         'count': 4,
6         'detailsURL': '',
7         'name': 'vEdge Hardware Health',
8         'status': 'error',
9         'statusList':
10          [
11            {
12              'count': 4,
13              'detailsURL': '/dataservice/device/hardwarehealth/detail?state=normal',
14              'message': '4 (normal=4, warning=0, error=0)',
15              'name': 'normal',
16              'status': 'up'
17            }
```

Refer to the exhibit. Cisco SD-WAN deployment must be troubleshooted using vManage APIs. A call to vEdge Hardware Health API returns the data in the exhibit (only a portion is shown). If this JSON is converted to a Python dictionary and assigned to the variable "d", how the status is accessed that is indicated on line 16?

A. d[data][0][statusList][0][status]

B. d['data']['statusList']['status']

C. d{'data'}[0]{'statusList'}[0]{'status'}

D. d['data'][0]['statusList'][0]['status']

ANSWER: B

Explanation:

`d['data']['statusList']['status']` correctly follows the nested JSON object hierarchy shown by the API response. After JSON is decoded into Python, each JSON object becomes a dictionary, so its members are accessed by key. The outer response dictionary held in `d` contains the `data` key. Its value is an object containing `statusList`, and that object in turn contains the `status` value indicated in the exhibit. Therefore, successive dictionary-key lookups retrieve the required status field without positional indexing. List indexes such as `[0]` are appropriate only when the corresponding JSON value is an array; the shown hierarchy for this field is object-based. The expression also uses ordinary Python string delimiters, which are required for executable Python source code. Cisco SD-WAN vManage APIs return structured JSON payloads, and clients commonly decode those payloads before traversing their nested data using the response schema. Cisco's SD-WAN API documentation is available at Cisco DevNet SD-WAN documentation. Python's documentation describes dictionary access by key in its mapping types reference.

QUESTION NO: 8

FILL BLANK

Information about a rebooted device needs to be displayed with an ID of 260faff9-2d31-4312-cf96-143b46db0211 using the Cisco SD-WAN vManage Administration APIs. The API documentation states that deviceId is a required request parameter. Fill in the blank to create the REST call.

https://vmanage-ip-address:8443/dataservice/device/action/reboot [ ] 260faff9-2d31-4312-cf96-143b46db0211

A. “deviceId”:

ANSWER: A

Explanation:

The required JSON request-property name is `"deviceId":`, using a lowercase `d` and an uppercase `I`. Cisco vManage REST APIs define this parameter with exact camel-case spelling. The device UUID, `260faff9-2d31-4312-cf96-143b46db0211`, is supplied as the string value associated with that property in the request payload. Therefore, the relevant portion of the payload is `{"deviceId":"260faff9-2d31-4312-cf96-143b46db0211"}`. JSON member names are case-sensitive in API payload processing, so preserving Cisco’s documented `deviceId` spelling is necessary for the reboot request to identify the intended device correctly. Cisco’s reboot-device API reference identifies `deviceId` as a required request parameter for this operation. See the Cisco vManage Reboot Device API reference and the Cisco Catalyst SD-WAN API documentation for the REST API conventions and payload definitions.

QUESTION NO: 9

Refer to the exhibit.

Which NETCONF statement type is represented by +--rw address* [ip]?

A. list

B. leaf-list

C. container

D. submodule

ANSWER: A

Explanation:

The correct statement type is **list**. This notation is generated by the YANG tree-diagram convention used to display a schema. In that convention, `+--rw` indicates a configuration data node with read-write access. The node name `address` is followed by an asterisk, `*`, which identifies it as a list that can have zero or more entries. The bracketed value, `[ip]`, identifies the list key: each `address` list entry is uniquely identified by its `ip` key leaf.

YANG lists represent repeatable collections of structured entries. Each entry may contain one or more child nodes, and the key ensures that a NETCONF client can address, create, modify, or delete a particular instance unambiguously. For example, a client could target the address entry whose key has a specified IP value rather than treating all address data as a single scalar value. This is the intended data-modeling structure when multiple address records are permitted and each record is distinguished by an IP identifier. The YANG language specification defines the `list` statement and its mandatory key behavior, while the YANG tree-diagram notation specifies the asterisk and bracketed-key display conventions. See RFC 7950: The YANG 1.1 Data Modeling Language and RFC 8340: YANG Tree Diagrams.

QUESTION NO: 10

Which two features are foundations of a software-defined network instead of a traditional network? (Choose two.)

A. control plane and data plane are tightly coupled

B. build upon a robust software stack

C. requires device by device-level configurations

D. automated through expressed intent to a software controller

E. requires significant physical hardware resources

ANSWER: B D

Explanation:

Software-defined networking is built upon a robust software stack because it abstracts network capabilities from the individual hardware devices and exposes them through software, APIs, and centralized management functions. This abstraction lets network teams define, deploy, and operate services consistently without treating each switch or router as an isolated configuration target. Cisco describes SDN as an architecture that separates network control and forwarding functions and makes the network directly programmable through software. See Cisco's Software-Defined Networking overview.

SDN is also automated through expressed intent to a software controller. Intent-based operation allows an administrator to state the desired business or network outcome, such as connectivity, segmentation, or policy, while the controller translates that intent into device-level configurations and continuously validates the intended state. Cisco DNA Center, for example, provides controller-based automation and policy-driven provisioning across the enterprise network. Cisco documents this controller-led approach in its Cisco Catalyst Center overview. Together, software abstraction and intent-driven controller automation are core SDN foundations.

QUESTION NO: 11

Which two types of solution are built with the Meraki Location Scanning API? (Choose two.)

A. networking automation

B. mapping

C. guest Wi-Fi

D. Sense

E. wayfinder

ANSWER: B E

Explanation:

The Meraki Location Scanning API supports location-aware applications by exposing observed Bluetooth Low Energy beacon information and associated location data from Meraki access points. This makes it suitable for **mapping** solutions, which present a floor-plan or indoor map and associate devices, assets, or points of interest with positions in that map. Such applications can use the API's scanning observations as an input for indoor-location workflows.

wayfinder solutions are also an intended use case. A wayfinding application builds on an indoor map to help a user identify a destination and navigate through a physical venue, such as an office, campus, retail site, or event space. Cisco's Meraki developer example demonstrates integrating Meraki location-scanning capabilities with Mapwize, an indoor-mapping and wayfinding platform. The integration illustrates how Meraki wireless infrastructure data can contribute to an application that displays locations and supports navigation within mapped spaces.

See Cisco's Mapwize wayfinding integration guide and the Meraki Dashboard API documentation for related Meraki wireless and Bluetooth API capabilities.

QUESTION NO: 12

Refer to the exhibit.

```
from ncclient import manager
with manager.connect(
      host='10.0.0.1',
      port=12022,
      username='cisco',
      password='cisco',
      hostkey_verify=False,
      allow_agent=False,
      look_for_keys=False,
      device_params={'name': 'iosxe'},
   ) as m:
```

Which ncclient method is used to collect the running configuration of a Cisco IOS XE device that uses NETCONF?

A. config=m.copy_config(source='running')

B. config=m.get(source='running')

C. config=m.collect_config(source='running')

D. config=m.get_config(source='running')

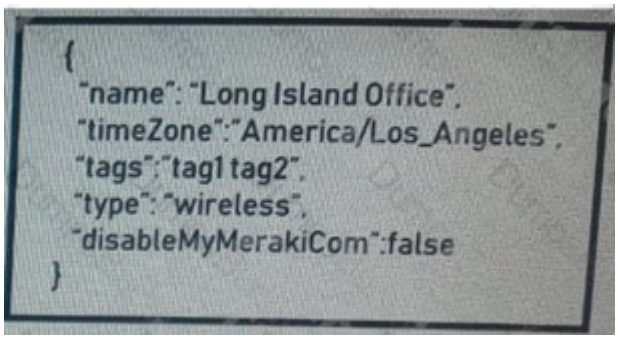
ANSWER: D

Explanation:

`config=m.get_config(source='running')` is the appropriate ncclient call because it invokes the NETCONF `<get-config>` operation and explicitly selects the running configuration datastore as its source. On Cisco IOS XE devices with NETCONF enabled, the running datastore represents the active configuration currently in use by the device. The returned reply is assigned to `config`, where its XML content can then be inspected or parsed by the automation script, commonly through properties such as `config.data_xml` or `config.data_ele`. ncclient exposes NETCONF operations through a Manager object, and `get_config()` accepts a datastore source, including `running`, for exactly this purpose. The NETCONF protocol defines `<get-config>` as the operation used to retrieve all or part of a specified configuration datastore. This makes the method suitable when the requirement is specifically to collect configuration data rather than operational state or to perform a configuration change. See the ncclient Manager API documentation and the IETF definition of the NETCONF get-config operation in RFC 6241.

QUESTION NO: 13

Refer to the Exhibit.



Which two parameters are mandatory when the Cisco Meraki API is used to create a network? (Choose two)

A. tags

B. name

C. time zone

D. type

E. disableMyMerakiCom

ANSWER: B D

Explanation:

`name` and `type` are the required inputs for the network-creation request shown in the exhibit. The `name` value supplies the human-readable network name that Meraki Dashboard displays and uses to identify the newly created network within the specified organization. The `type` value establishes the network's functional scope, such as appliance, switch, wireless, systems manager, camera, or combined network capabilities, so Meraki can create the appropriate network object and enable the applicable Dashboard services. Together, these values define both the identity and intended platform role of the new network; therefore, they must be present when using the API schema represented by the question. Cisco's Meraki API documentation describes organization-scoped network creation and the attributes used in request bodies. For current API versions, Cisco has evolved the corresponding network-capability field to `productTypes`; this question's `type` terminology reflects the API schema displayed in the exhibit. See the Cisco Meraki API network creation reference and the Cisco Meraki Dashboard API documentation.

QUESTION NO: 14

A programmer is creating a Meraki webhook Python script to send a message to Webex Teams. Which two elements should be configured to create this script? (Choose two.)

A. gRPC credentials

B. Webex Teams access token

C. XML formatted request

D. user authentication count

E. webhook server secret

ANSWER: B E

Explanation:

A **Webex Teams access token** is required because the script must authenticate to the Webex messaging API when it creates and sends a message. The token is supplied as a Bearer token in the authorization header for API calls, allowing the application to act with the permissions granted to the token owner. The script also needs the **webhook server secret** configured for the Meraki webhook receiver. Meraki supports a shared secret on webhook configurations and signs webhook requests so that the receiving application can verify that an incoming notification genuinely originated from Meraki before processing it and relaying an alert to Webex Teams. In a secure Python integration, the receiver validates the Meraki request signature using this configured secret, extracts the relevant event information from the JSON webhook payload, and then uses the Webex access token to submit the resulting message. This separates verification of the event source from authorization to post into Webex, which are both necessary integration controls. Cisco Meraki documents webhook shared-secret signing and validation in its webhook security documentation. Webex documents Bearer-token authentication for its REST APIs in the Webex API getting-started guide.

QUESTION NO: 15

What are two characteristics of RPC API calls? (Choose two.)

A. They can be used only on network devices.

B. They use only UDP for communications.

C. Parameters can be passed to the calls.

D. They must use SSL/TLS.

E. They call a single function or service.

ANSWER: C E

Explanation:

RPC (Remote Procedure Call) is an API interaction model in which a client invokes a specific procedure, method, or service operation that executes on a remote system. Therefore, “They call a single function or service.” is a core RPC characteristic: the request is directed to a defined operation rather than being primarily modeled as manipulation of a resource collection. The client commonly uses a generated or defined interface that identifies the remote procedure to invoke.

“Parameters can be passed to the calls.” is also fundamental to RPC. A client supplies the input arguments required by the remote procedure, which are serialized into a request message, transmitted to the server, deserialized there, and used during execution. The procedure can then return a result or response to the caller. This model allows a remote operation to resemble a local function call while handling the network communication and data encoding underneath the API contract.

RPC is a general distributed-computing approach, not a device-specific mechanism. Its transport and security depend on the RPC implementation and deployment requirements. For background on the RPC model, see the Open Group RPC specification and the gRPC core concepts documentation.

QUESTION NO: 16 - (SIMULATION)

Fill in the blank to complete the statement.

ANSWER: See the explanation for the answer

Explanation:

Answer: `DHCP`

Zero Touch Provisioning (ZTP) starts by using DHCP to obtain network addressing information and provisioning details, such as the location of the server or boot file used to retrieve the initial configuration/script.

QUESTION NO: 17 - (DRAG DROP)

DRAG DROP

```
GET: https://dnacsrv/api/v1/network-device
{
  "response": [
      {
        "type": "Cisco Catalyst 9300 switch",
        "errorCode": null,
        "family": "Switches and Hubs",
        "location": DC1,
        "role": "ACCESS",
        "macAddress: "a1:2b:30:40:41:50",
        "hostname": "cat_9k_1",
        "serialNumber": "FCW2136L0AK",
        "softwareVersion": "16.6.1",
        "locationName": null,
        "upTime": "13 days, 18:30:33.81",
        "softwareType": "IOS-XE",
        "collectionStatus": "Managed",
        "managementIpAddress": "10.10.22.66",
        "platformId": "C9300-24UX",
        "reachabilityStatus": "Reachable",
        "series": "Cisco Catalyst 9300 Series Switches",
        "snmpContact": "",
        "snmpLocation": "",
```

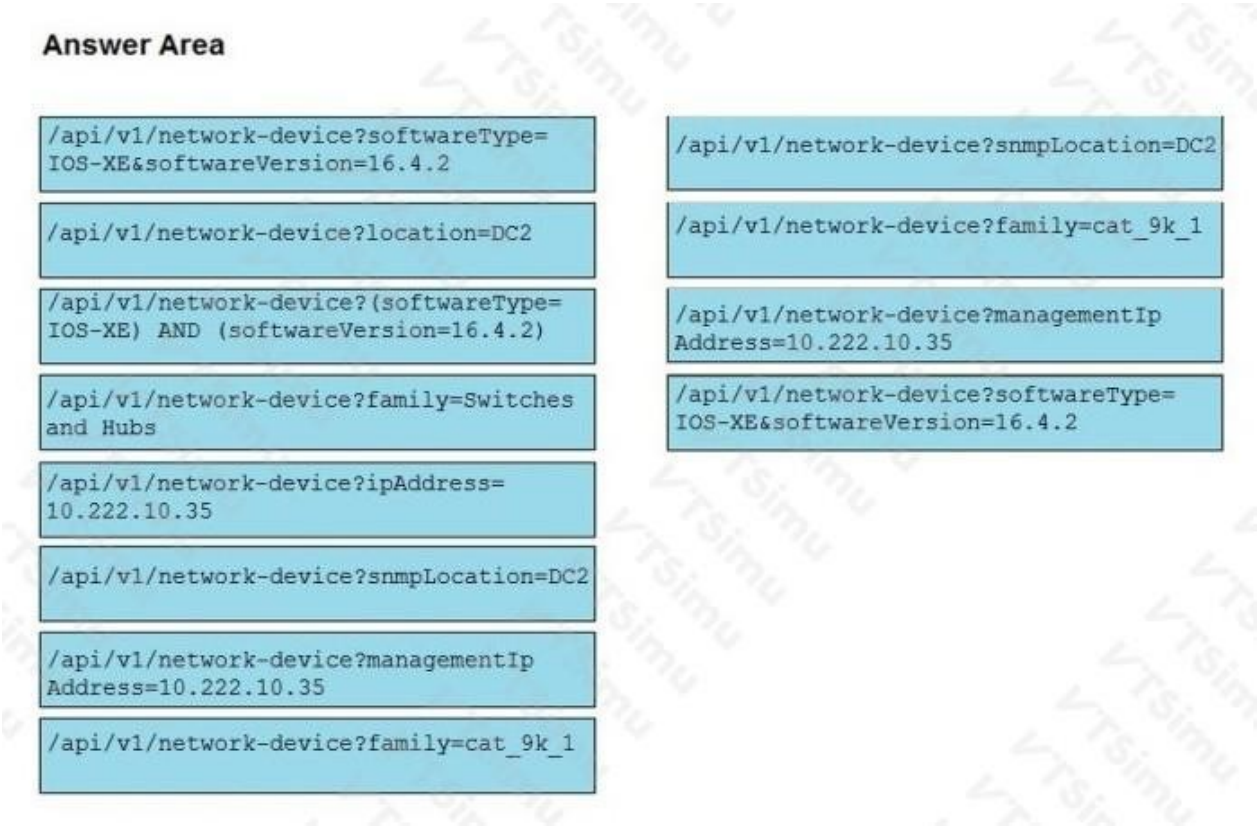
Refer to the exhibit. A GET request is issued to the Cisco DNA Center REST API. Drag and drop the GET request URL subpaths from the left onto the objectives on the right. Not all options are used.

Select and Place:

Answer Area

/api/v1/network-device?softwareType=IOS-XE&softwareVersion=16.4.2	List devices that are configured by using SNMP to be in the DC2 location
/api/v1/network-device?location=DC2	List device types
/api/v1/network-device?(softwareType=IOS-XE) AND (softwareVersion=16.4.2)	List the device that has an IP address of 10.222.10.35
/api/v1/network-device?family=Switches and Hubs	Display Cisco IOS XE devices that have IOS version 16.4.2
/api/v1/network-device?ipAddress=10.222.10.35	
/api/v1/network-device?snmpLocation=DC2	
/api/v1/network-device?managementIpAddress=10.222.10.35	
/api/v1/network-device?family=cat_9k_1	

ANSWER:



Explanation:

Cisco DNA Center's network-device collection supports query-string filtering, so each objective is fulfilled by selecting the parameter that represents the device attribute named in that objective. For devices configured with the SNMP location value `DC2`, the request uses `snmpLocation=DC2`. This directs the API to match the SNMP location configured on the inventory device record, rather than a general location field. For listing a particular device type, the applicable inventory classification is the device `family`; therefore, the request containing `family=cat_9k_1` is the appropriate device-family filter.

A device's management address is queried through the `managementIpAddress` parameter. Supplying `managementIpAddress=10.222.10.35` returns the inventory device associated with that management IP address. Finally, the IOS XE and release requirement needs both characteristics in the same request: `softwareType=IOS-XE` identifies the operating-system type and `softwareVersion=16.4.2` identifies the required release. Combining those filters with `AND` limits the result to devices satisfying both conditions. These filters follow the Cisco DNA Center Intent API model for retrieving and narrowing network-device inventory results. Cisco documents the network-device retrieval resource and its supported query parameters in the Cisco DNA Center Get Network Device List API documentation. Additional API guidance and current platform documentation are available through the Cisco DNA Center developer documentation.

QUESTION NO: 18

Which two actions do Python virtual environments allow users to perform? (Choose two.)

A. Simplify the CI/CD pipeline when checking a project into a version control system, such as Git.

B. Efficiently port code between different languages, such as JavaScript and Python.

C. Run and simulate other operating systems within a development environment.

D. Quickly create any Python environment for testing and debugging purposes.

E. Quickly create an isolated Python environment with module dependencies.

ANSWER: D E

Explanation:

Python virtual environments are lightweight, self-contained Python installations created for an individual project or task. They allow developers to quickly create an isolated Python environment with module dependencies, so packages installed for one project do not alter or conflict with packages used by another project or with the system-wide Python installation. Each environment has its own interpreter context and package-installation location, enabling a project to use the specific dependency versions it requires.

They also allow developers to quickly create a Python environment for testing and debugging purposes. A developer can create a fresh environment, install only the packages needed to reproduce an issue or test a change, and then discard and recreate that environment as needed. This improves repeatability and makes it easier to validate an application against controlled dependency sets. Python's standard `venv` module is specifically intended to create these isolated environments, while tools such as `pip` install project dependencies into the active environment. See the Python documentation for venv — Creation of virtual environments and the Python Packaging User Guide's guide to installing packages using virtual environments.

QUESTION NO: 19

Which curl command is used to update the SNMP community of network ID "1234567" to read-only?

A.

```
curl -L -H 'X-Cisco-Meraki-API-Key: <key>' \
  -H 'Content-Type: application/json' \
  -X PUT --data-binary '{ \
    "access":"users", \
          "communityString":"readonly"}' \
          'https://api.meraki.com/api/v0/networks/1234567/snmpSettings'
```

B.

```
curl -L -H 'X-Cisco-Meraki-API-Key: <key>' \
  -H 'Content-Type: application/json' \
  -X PUT --data-binary '{ \
    "access":"community", \
          "communityString":"readonly"}' \
          'https://api.meraki.com/api/v0/networks/1234567/snmpSettings'
```

C.

```
curl -L -H 'X-Cisco-Meraki-API-Key: <key>' \
  -H 'Content-Type: application/json' \
  -X PUT --data-binary '{ \
    "access":"users", \
          "usersname":"snmp", \
          "passphase":"readonly"}' \
          'https://api.meraki.com/api/v0/networks/1234567/snmpSettings'
```

D.

```
curl -L -H 'X-Cisco-Meraki-API-Key: <key>' \
  -H 'Content-Type: application/json' \
  -X POST --data-binary '{ \
    "access":"community", \
          "communityString":"readonly"}' \
          'https://api.meraki.com/api/v0/networks/1234567/snmpSettings'
```

A. Option A

B. curl --request PUT --url "https://api.meraki.com/api/v1/networks/1234567/snmp" --header "Authorization: Bearer " --header "Content-Type: application/json" --data '{"access":"community","communityString":""}'

C. Option C

D. Option D

ANSWER: B

Explanation:

`curl --request PUT --url "https://api.meraki.com/api/v1/networks/1234567/snmp" --header "Authorization: Bearer <API_KEY>" --header "Content-Type: application/json" --data '{"access":"community","communityString":"<COMMUNITY_STRING>"}'` is correct because it invokes the Meraki Dashboard API operation that updates SNMP settings for one specific network. The network identifier is placed in the request path, and `PUT` is the HTTP method defined for changing the existing network SNMP configuration rather than retrieving it. The JSON request body selects `community` access and supplies the community string, which configures SNMP community-based access. On Meraki-managed devices, SNMP community access is used for SNMP version two community polling and is read-only; it supports monitoring data collection rather than SNMP write operations. The request must also include Dashboard API authentication and a JSON content type so that the service can authenticate the caller and parse the submitted settings. Cisco's endpoint reference documents the network SNMP update path, the `PUT` method, and the accepted `access` and `communityString` fields. See Cisco Meraki API: Update Network SNMP and Cisco Meraki Dashboard API Getting Started.

QUESTION NO: 20

What are two benefits of leveraging Ansible for automation of Cisco IOS XE Software? (Choose two.)

A. Ansible playbooks are packaged and installed on IOS XE devices for automatic execution when an IOS device reboots.

B. All IOS XE operating systems include Ansible playbooks for basic system administration tasks.

C. It is a device-independent method for automation and can be used with any type of device or operating system.

D. Ansible playbooks can be written from the IOS XE EXEC command line to configure the device itself.

E. It does not require any modules of software except SSH to be loaded on the network device.

ANSWER: C E

Explanation:

Ansible provides agentless automation: the control node connects to Cisco IOS XE devices using standard management connectivity, commonly SSH, rather than requiring an Ansible agent or an additional software package to be installed and maintained on every managed device. This reduces operational overhead and makes it easier to begin automating an existing network. IOS XE can be managed through Ansible network connection methods and Cisco-supported collections, while the playbook logic remains on the automation control node.

Ansible is also designed as a broadly device-independent automation framework. The same YAML-based playbook structure, inventory model, variables, and task workflow can be used across network platforms, servers, cloud services, and other supported endpoints. Platform-specific functionality is supplied through collections and modules, allowing a consistent automation approach while still using the appropriate IOS XE capabilities. This portability helps teams standardize automation practices across heterogeneous environments rather than creating entirely separate automation systems for each device type.

See Cisco's Ansible overview for IOS XE and the Ansible Network Automation documentation.