DUMPSBOSS.

Automating and Programming Cisco Service Provider Solutions (300-535 SPAUTO)

Cisco 300-535

Version Demo

Total Demo Questions: 8

Total Premium Questions: 87

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Software Development and Design	19
Topic 2, APIs	27
Topic 3, Cisco IOS XE Programmability	4
Topic 4, Cisco IOS XR Programmability	13
Topic 5, Cisco NSO Programmability	17
Topic 6, Model-Driven Telemetry	5
Topic 7, Mix Questions	2
Total	87

QUESTION NO: 1

Refer to the exhibit.

```
<?xml version= “1.0” encoding= “UTF-8” ?>
<employees>
        <employee>
                <id>1</id>
                <firstName>Jane</firstName>
                <firstName>Doe</lastName>
        </employee>
        <employee>
                <id>2</id>
                <firstName>Tim</firstName>
                <lastName>Public</lastName>
        </employee>
        <employee>
                <id>3</id>
                <firstName>John</firstName
                <lastName>Doe</latName>
        </employee>
</employees>
```

Using the provided XML snippet, which Xpath expression prints “Jane”?

A. //employee[1]/firstName/value()

B. //employee[0]/firstName/value()

C. //employee[1]/firstName/text()

D. //employee[0]/firstName/text()

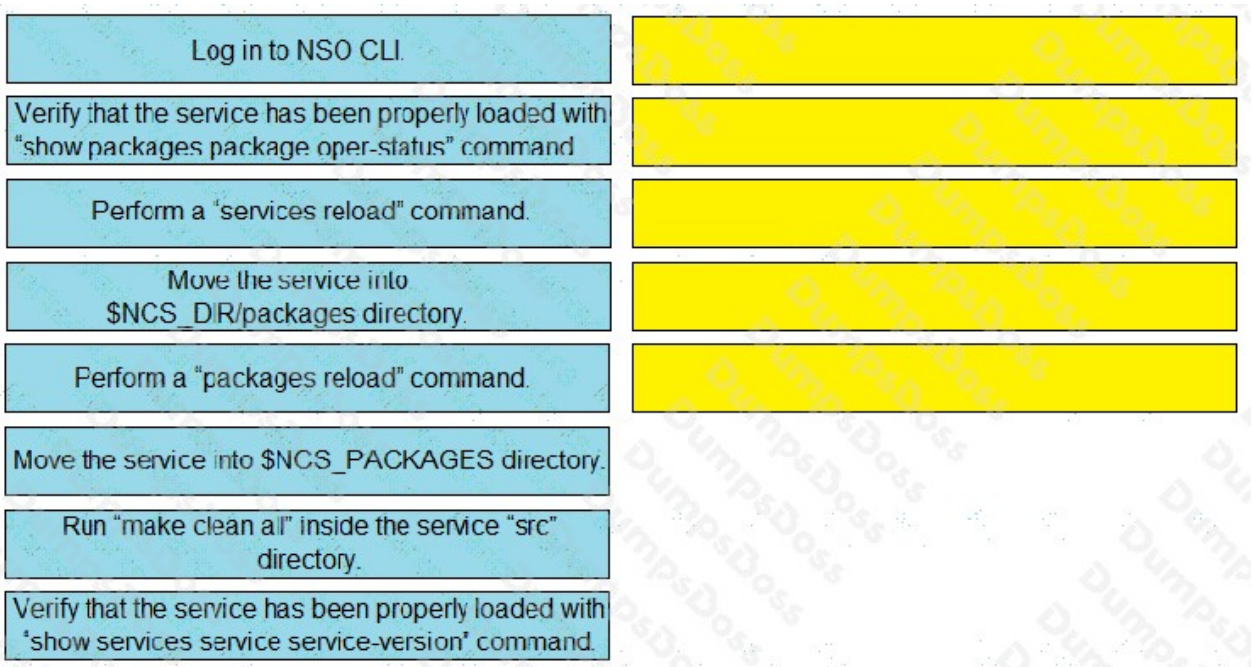
ANSWER: C

Explanation:

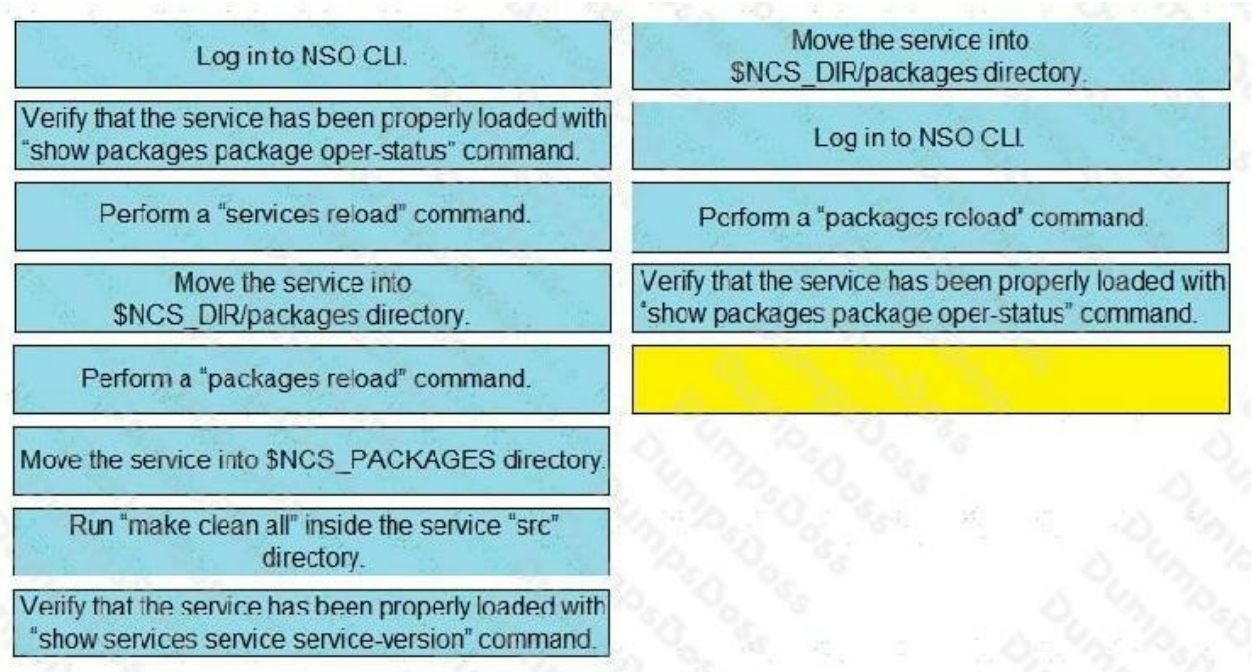
`//employee[1]/firstName/text()` selects the text node containing “Jane.” In XPath, location steps navigate an XML document element by element. The descendant-or-self path abbreviation `//` locates the relevant `employee` element in the supplied XML structure, and the predicate `[1]` uses XPath’s one-based positional indexing to select the first matching employee in that context. The next step, `/firstName`, selects that employee’s `firstName` child element. Finally, `text()` selects the element’s text-node child, which is the character data “Jane”; an XPath processor can then serialize that selected text node as the displayed value. XPath distinguishes element nodes from their text-node contents, so using `text()` is the direct node test for retrieving the string stored between the opening and closing `firstName` tags. The XPath specification defines predicates as filters on node sequences and specifies that positional predicates use positions beginning at one. It also defines `text()` as the node test that matches text nodes. See the W3C XPath specification and the MDN XPath text() reference.

QUESTION NO: 2 - (DRAG DROP)

Drag and drop the steps from the left into the correct order on the right to deploy an already created service into NSO. Not all options are used.



ANSWER:



Explanation:

To deploy an already-created NSO service, start by placing the complete service package in the `$NCS_DIR/packages` directory. This is the package location NSO scans when it loads package content. The directory must contain the package structure that was previously built, including its package metadata and service implementation files, so NSO can recognize it as an installable package.

After the package is in place, log in to the NSO CLI and run `packages reload`. This command tells the running NSO instance to rescan its configured package locations and load the newly added package without requiring a full NSO restart. During this process, NSO reads the package definition, loads its YANG models and templates, and initializes the service package for use by NSO.

Finally, use `show packages package oper-status` to confirm that the package was successfully loaded and that its operational status is healthy. This validation is important because it verifies NSO accepted the package and made it available to the service framework. These are deployment actions for an existing package; the package build and source-

cleaning workflow is not part of loading an already-created service into NSO. Cisco's service-package lab material describes the package placement, reload, and operational-status validation workflow in the Cisco Live NSO service package lab guide.

QUESTION NO: 3

Which NETCONF datastore is locked while the network device configuration is edited?

A. running

B. common

C. startup

D. working

ANSWER: A

Explanation:

The **running** datastore is the correct answer in the direct NETCONF configuration workflow used when edits are applied to the active device configuration. NETCONF uses the `<lock>` operation to obtain exclusive access to a datastore before making changes, preventing another management client from concurrently changing that same configuration. A client locks the running datastore, performs one or more `<edit-config>` operations with running as the target, and then releases the lock with `<unlock>`. Because running represents the device's current operational configuration, this approach protects the active configuration from conflicting edits during the transaction. The NETCONF base specification defines locking as a datastore-level mechanism and illustrates a lock request whose target is `running`. Cisco's NETCONF documentation likewise identifies running as the active configuration datastore exposed for configuration operations on supported platforms. See the NETCONF protocol specification in RFC 6241 and Cisco's NETCONF Programmability Configuration Guide.

QUESTION NO: 4

Which two data formats are human readable? (Choose two.)

A. YAML

B. Apache Arrow

C. gRPC

D. binary

E. JSON

ANSWER: A E

Explanation:

YAML and JSON are human-readable, text-based data-serialization formats commonly used to represent structured configuration and API data. YAML is designed for readability: it uses indentation and simple key-value notation, allowing people to write and review hierarchical data without extensive punctuation. JSON represents objects and arrays in plain Unicode text using explicit structural characters such as braces, brackets, commas, and quotation marks. Although software normally parses both formats, their contents can be directly read, edited, and understood in a text editor, which makes them appropriate for configuration files, payloads, and automation workflows. The YAML specification describes YAML as a human-friendly data-serialization language, while the JSON standard defines JSON as a text format for structured data interchange. In Cisco automation contexts, JSON is widely used in REST API requests and responses, and YAML is frequently used in declarative automation and inventory/configuration tooling. References: YAML 1.2.2 Specification and RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format.

QUESTION NO: 5

```
#!/usr/bin/env python

from ydk.models.openconfig.openconfig_interfaces import Interfaces
from ydk.errors import YError

def read_interfaces(crud_service, provider):

    intf_f = Interfaces()

    try:
        interfaces = crud_service.read(provider, intf_f)
        for interface in interfaces.interface:
            print(interface.name)
    except YError:
        print('An error occurred.')
```

Refer to the exhibit. When YDK is used to interact with Cisco routers, what is the purpose of passing intf_f into the crud_service.read() method?

A. The Interfaces() class acts as a NETCONF filter, which limits the data returned to that of the openconfig:interfaces YANG model.

B. It provides the data types of the openconfig:interfaces model to the router for dynamic configuration of the interfaces.

C. It locks the interfaces from modification by other active NETCONF sessions.

D. It passes default values into the crud_service, which reconfigures all interfaces to their default configurations.

ANSWER: A

Explanation:

The Interfaces() class acts as a NETCONF filter, which limits the data returned to that of the openconfig:interfaces YANG model. In YDK, `crud_service.read()` accepts an entity object that represents the requested YANG data subtree. Passing `intf_f`, an instance of the OpenConfig `Interfaces` container, tells YDK to construct a read request scoped to that model's interface hierarchy rather than retrieving unrelated configuration and operational data from the device. The values populated on the entity can further narrow the request when list keys or child elements are specified; when the top-level Interfaces container is used as the filter, the request targets the available interface data under that container. YDK then decodes the reply into the corresponding Python model objects, allowing the application to work with strongly modeled OpenConfig interface data. This entity-based filtering approach is central to YDK CRUD operations and maps the model object hierarchy to the data requested through the underlying device protocol, such as NETCONF. See the YDK CRUDService API documentation and Cisco's IOS XE programmability documentation for NETCONF and YANG model usage.

QUESTION NO: 6 - (SIMULATION)

Fill in the blank to complete the statement about NETCONF and Python libraries.

_____is a Python library that facilitates client-side scripting and deploying changes to the network using the NETCONF protocol.

ANSWER: See the explanation for the answer

Explanation:

Answer: `ncclient`

`ncclient` is a Python library for client-side NETCONF scripting. It provides NETCONF operations such as connecting to a device, retrieving configuration, editing configuration, locking candidate/running configuration, and committing changes.

```
from ncclient import manager

with manager.connect(host="router.example.com", port=830,
                     username="admin", password="password",
                     hostkey_verify=False) as m:
    print(m.get_config(source="running"))
```

Reference: https://pypi.org/project/ncclient/

QUESTION NO: 7

Refer to the exhibit.

Which two URI entries are optional and functional in this RESTCONF URI structure? (Choose two.)

A. fragment

B. query

C. operation

D. api-entry

E. path

ANSWER: B E

Explanation:

In RESTCONF, the **path** identifies a specific resource below a RESTCONF resource collection, but it can be omitted when the request is addressed to the relevant collection or root resource itself. For example, a request can address the RESTCONF data resource without appending a data-resource path, while a more targeted request can include a path to a particular YANG data node or list instance. RESTCONF defines this resource identifier portion as optional in the URI structure.

The **query** component is also optional and functional. RESTCONF uses query parameters to modify request processing or the representation returned by the server. Common uses include selecting configuration versus nonconfiguration content, choosing a response depth, applying field selection, or controlling pagination where supported. Query parameters follow the resource URI after a question mark and are interpreted by the RESTCONF server as request parameters. This allows clients to request a tailored response without changing the underlying resource path.

These optional components are defined in RESTCONF URI and request-processing behavior in RFC 8040, section 3.5 and its query-parameter definitions.

QUESTION NO: 8

The Netmiko BaseConnection class contains a method called “send_config_set()”. Which two actions does this method perform on the device? (Choose two.)

A. It takes a filename parameter that executes commands contained in that file on the device.

B. It requires the user to explicitly send configure terminal and exit commands to the device to enter and exit configuration mode.

C. It automatically enters and exits configuration mode on the device.

D. It takes a Python iterable, such as a list of commands, and executes them in order on the device.

E. It saves the running configuration to the startup configuration after executing the configuration commands on the device.

ANSWER: C D

Explanation:

`send_config_set()` is designed to apply a set of configuration commands as a single workflow. It accepts an iterable of command strings, which commonly is a Python list, and sends those commands to the network device sequentially. This makes it appropriate for automation tasks in which several related configuration lines must be applied in their intended order, such as configuring an interface followed by its addressing and routing attributes.

By default, the method also handles configuration-mode transitions for platforms that use a distinct configuration mode. Netmiko checks whether the session is already in configuration mode, enters it when necessary, sends the supplied configuration commands, and exits configuration mode when complete. Its documented parameters include `enter_config_mode` and `exit_config_mode`, both of which default to enabled, confirming this built-in behavior. The method returns the device output generated while processing the configuration set; saving the resulting running configuration is a separate operation that must be requested through an appropriate device command or workflow. See the Netmiko BaseConnection.send_config_set API documentation and the Netmiko project documentation.