PassQueen
a.k.a dumpsqueen.com

Automation and DevOps Associate (JNCIA-DevOps)

Juniper JN0-223

Version Demo

Total Demo Questions: 6

Total Premium Questions: 63

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, DevOps Concepts	7
Topic 2, DevOps Tools	1
Topic 3, Automation Tools	15
Topic 4, Junos Automation and Programmability	40
Total	63

QUESTION NO: 1

YAML uses which two data structures? (Choose two.)

A. arrays

B. mappings

C. sequences

D. objects

ANSWER: B C

Explanation:

YAML's fundamental collection data structures are mappings and sequences. A mapping represents an unordered association of unique keys with values, comparable to a dictionary or hash table. In block-style YAML, a mapping is commonly written with a colon between each key and its associated value, such as `interface: ge-0/0/0`. Mappings are especially useful in automation data because they describe named attributes, configuration parameters, and structured device facts.

A sequence represents an ordered collection of zero or more items. Block-style sequences normally use a hyphen before each entry, for example `- router1` and `- router2`. Sequences are used when automation inputs must retain an explicit item order, such as a list of tasks, interfaces, addresses, or target devices. YAML documents may nest mappings and sequences to model more complex configuration and operational data. The YAML specification identifies mappings and sequences as its collection types; scalar values, such as strings and numbers, can be used as the individual values within those collections. See the YAML 1.2.2 specification and the official YAML project site.

QUESTION NO: 2

You must use Junos PyEZ to configure unique IP addresses on individual machines.

Which two features willpermit this requirement? (Choose). Ian SCP module

A. an SCP module

B. a BSON data file

C. a YAML data file

D. a Jinja2 template

ANSWER: C D

Explanation:

A YAML data file and a Jinja2 template together support repeatable but device-specific configuration generation with Junos PyEZ. YAML provides a structured, readable way to define variables for each managed device, such as a hostname, interface name, address, and prefix length. A script can select or load the appropriate set of values for an individual machine before generating its configuration.

A Jinja2 template supplies the reusable Junos configuration structure and inserts the device-specific YAML values at render time. For example, one template can contain an interface and address-family hierarchy while variables render a distinct IP address for each target device. The resulting configuration text can then be loaded through Junos PyEZ configuration utilities, allowing the same automation workflow to safely configure many devices without manually maintaining separate static configuration snippets.

This data-and-template approach separates variable inventory data from configuration logic, which improves consistency, reviewability, and reuse in automation pipelines. Juniper documents Jinja2 templates as a supported method for generating and loading configuration with PyEZ; YAML is commonly used as the corresponding source of template variables. See the Junos PyEZ configuration documentation and the Jinja template documentation.

QUESTION NO: 3

You are asked to use the REST API to retrieve interface configuration information from your Junos device. You decide to use a cURL HTTP GET command to retrieve this information.

In this scenario, which two statements are correct? (Choose two.)

A. You can retrieve this data in HTML or JSON formats.

B. You must have SSH enabled on the Junos device.

C. You can retrieve this data in XML or JSON formats.

D. You must Include the authentication Information with each request.

ANSWER: C D

Explanation:

“You can retrieve this data in XML or JSON formats.” is correct because the Junos REST API represents resources using standard structured data formats. A client such as cURL can request the desired representation by supplying an appropriate HTTP `Accept` header, such as `application/xml` or `application/json`. This enables interface configuration data to be consumed directly by automation tools and scripts, rather than requiring screen-oriented output.

“You must Include the authentication Information with each request.” is also correct. REST requests to a Junos device must be authenticated before protected configuration information can be accessed. With cURL, credentials can be supplied using HTTP basic authentication, for example with the `-u` option, or a client can use the authentication mechanism and session/token handling supported by the configured REST API service. The request must therefore carry valid authentication context when retrieving the resource. Junos REST API documentation describes REST resource access and supported request/response representations, while the cURL examples show how HTTP authentication is provided by clients. See the Junos REST API overview and Juniper’s REST API request documentation.

QUESTION NO: 4

You want to perform a dry run on the myPlays playbook and use a custom inventory file called myRouters.ini.

Which Ansible command would you use in this scenario?

A. ansible-playbook myPlays --check -i myRouters.ini

B. ansible-playbook myPlays —extra-vars "inventory_file=myRouters .ini"

C. ansible-playbook myPlays —extra-vars "dry run=True" myRouters.ini

D. ansible-playbook myPlays —limit myRouters

ANSWER: A

Explanation:

`ansible-playbook myPlays --check -i myRouters.ini` is the appropriate command because it combines the two required Ansible playbook command-line controls. The `--check` flag enables check mode, commonly called a dry run. In check mode, Ansible evaluates tasks and reports the changes it would ordinarily attempt, without making those changes to managed hosts, provided that the modules used support check mode. This is useful for validating the expected effect of a playbook before applying it in an operational environment. The `-i myRouters.ini` argument explicitly selects `myRouters.ini` as the inventory source, allowing the playbook to target the hosts and groups defined in that custom inventory rather than the default inventory. The playbook name follows `ansible-playbook`, and both options can be supplied in either order. For authoritative command-line documentation, see the Ansible playbook CLI reference. Further details about check mode and its module support are available in the Ansible check mode documentation.

QUESTION NO: 5

What is the default port for NETCONF connections over SSH?

A. 22

B. 8080

C. 830

D. 433

ANSWER: C

Explanation:

830 is the well-known, IANA-assigned TCP port for NETCONF sessions transported over SSH. NETCONF uses SSH as a secure transport for management and configuration operations, providing authentication, confidentiality, and integrity for communications between a NETCONF client and a Junos device. When a client initiates a standard NETCONF-over-SSH connection, it connects to TCP port 830 and requests the SSH subsystem named `netconf`. This port assignment is defined by the NETCONF over SSH specification and is used by Junos OS for its NETCONF SSH service by default. A device can be configured to use a different port when operational requirements require it, but 830 remains the default and standards-based port expected for NETCONF over SSH. See Juniper's NETCONF SSH connection documentation and RFC 6242, Using the NETCONF Protocol over Secure Shell (SSH).

QUESTION NO: 6

What is the correct sequence for Python script execution?

A. The code is translated to byte code, the byte code is executed in runtime, and then the code is interpreted.

B. The code is interpreted, the code is translated to byte code, and then the byte code is executed in runtime.

C. The code is translated to byte code, the code is interpreted, and then the byte code is executed in runtime.

D. The byte code is executed in runtime, the code is interpreted, and then the code is translated to byte code.

E. The source code is compiled to bytecode, and then the bytecode is executed by the Python virtual machine.

ANSWER: E

Explanation:

The correct sequence is: "The source code is compiled to bytecode, and then the bytecode is executed by the Python virtual machine." When a Python implementation such as CPython runs a script, it first reads and parses the source text, then compiles the parsed code into an intermediate representation called bytecode. This bytecode is the instruction format used by the Python virtual machine. The runtime then executes those bytecode instructions within execution frames, performing operations such as name lookup, function calls, and object manipulation. In CPython, a cached bytecode file may also be written for importable modules when appropriate, but that cache is an optimization and does not alter the fundamental compile-then-execute model.

Calling the initial stage "interpreting the source code" before bytecode translation is imprecise. Python is commonly described as an interpreted language because its virtual machine executes bytecode rather than directly producing a standalone native executable, but source code must be compiled into bytecode before that virtual-machine execution occurs. The Python language reference describes execution in terms of code blocks and execution frames, while CPython's compiler documentation describes compilation of source constructs into executable bytecode. See the Python execution model and the CPython compiler documentation.