DUMPSARENA

Red Hat Certified Specialist in Ansible Automation exam

RedHat EX407

Version Demo

Total Demo Questions: 17

Total Premium Questions: 172

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Understand and use essential components of Ansible	35
Topic 2, Install and configure an Ansible control node	12
Topic 3, Configure Ansible managed nodes	4
Topic 4, Write Ansible Playbooks	28
Topic 5, Configure and use Ansible roles	27
Topic 6, Configure and manage variables	27
Topic 7, Configure task control	20
Topic 8, Use provided documentation to look up specific information about Ansible modules and commands	18
Topic 9, Mix Questions	1
Total	172

QUESTION NO: 1

What are the default number of forks in Ansible?

A. 1

B. 5

C. 50

D. 10

ANSWER: B

Explanation:

The default number of forks in Ansible is 5. A fork is a parallel worker process used by the Ansible controller to execute tasks against managed hosts concurrently. Therefore, with the default configuration, Ansible can normally process up to five target hosts at the same time for a task, subject to factors such as the play's serial setting, inventory limits, task throttling, and the hosts available in the current batch.

This controller-side default is defined by the `forks` configuration setting. It can be changed globally in `ansible.cfg`, through the `ANSIBLE_FORKS` environment variable, or for an individual command with the `--forks` (or `-f`) command-line option. Increasing the value can improve throughput for large inventories when the control node and network have sufficient capacity, but it should be selected according to the controller's available CPU, memory, connection capacity, and the managed systems' ability to handle concurrent work.

Red Hat's Ansible configuration reference lists `DEFAULT_FORKS`, represented by the `forks` setting, with a default value of `5`. See the Ansible configuration reference for DEFAULT_FORKS and the ansible-playbook command reference.

QUESTION NO: 2

State whether the following statement is true or false.

If you create your own ansible facts file, it can be executable.

A. True

B. False

ANSWER: A

Explanation:

True is correct. Ansible supports local facts, which are custom facts stored on managed hosts in the directory defined by `fact_path`; on Linux and other POSIX systems, the default directory is `/etc/ansible/facts.d`. A local fact file using the `.fact` extension can be either a static INI-formatted or JSON-formatted file, or an executable script. When the file is executable, Ansible runs it during fact gathering and reads the script's standard output as the fact data. The script must emit valid JSON or INI data so Ansible can add the resulting values beneath the `ansible_local` fact namespace. This makes executable local facts useful when a value must be calculated dynamically on the managed host, such as discovering an application version, querying a locally available command, or deriving configuration state. If a local fact is created or changed during a play, rerunning the `setup` module can refresh the facts immediately in the same play. See the Ansible documentation on adding custom facts and the ansible.builtin.setup module.

QUESTION NO: 3

Examine the following inventory excerpt file named /home/user/ansible/inventory. [dbservers] db1.example.com

Which of the following files does Ansible check for variables related to that inventory? (Choose all that apply.)

A. /home/user/ansible/dbservers

B. /home/user/ansible/host_vars/db1.example.com

C. /home/user/ansible/host_vars/db1

D. /home/user/ansible/group_vars/dbservers

ANSWER: B D

Explanation:

Ansible supports variable files organized by inventory host and group. When an inventory source is located at `/home/user/ansible/inventory`, Ansible searches the inventory directory for the conventional `host_vars` and `group_vars` directories. The host entry is named `db1.example.com`, so `/home/user/ansible/host_vars/db1.example.com` is a valid host-variable file location. Variables defined there apply specifically to that inventory host. Ansible also recognizes the `dbservers` inventory group and loads variables from `/home/user/ansible/group_vars/dbservers`. Those variables apply to every host that belongs to the `dbservers` group, including `db1.example.com`. These paths may be files without extensions or directories containing supported variable files, such as YAML files. This inventory-relative layout is the standard approach for keeping host-specific and group-wide data separate while allowing Ansible to discover it automatically during inventory processing. See the official Ansible documentation on organizing host and group variables and inventory behavior.

QUESTION NO: 4

Which of the following are valid privilege-escalation directives in a playbook? (Choose all that apply.)

A. become

B. become_user

C. become_method

D. become_flags

E. sudo_user

ANSWER: A B C D

Explanation:

The `become` directive enables privilege escalation for a play, block, role, or task. The `become_user` directive selects the user to become, normally `root` when no other user is specified. The `become_method` directive selects the escalation method, such as `sudo`, and `become_flags` provides flags for the selected method when required.

`sudo_user` is not a current general privilege-escalation keyword. Ansible uses `become_user` instead. See the Ansible privilege escalation guide and the playbook keyword reference.

QUESTION NO: 5

An Ansible inventory may contain which of the following? (Choose all that apply.)

A. Variable declarations

B. Groups

C. Hosts

D. Plays

ANSWER: A B C

Explanation:

An Ansible inventory defines the managed nodes that Ansible can target, so it can contain hosts, either as individual host entries or as members of groups. Hosts may be listed by DNS name, IP address, or another inventory hostname, and Ansible uses the inventory to determine the systems available for a playbook or ad hoc command.

Inventories can also contain groups. A group is a named collection of hosts that lets automation target related systems together, such as web servers or database servers. Ansible supports nested groups as well, allowing a parent group to include other groups and making it possible to model larger environments hierarchically.

Variable declarations are also valid inventory content. Variables can be assigned to individual hosts, to groups, or to all hosts, either directly in inventory data or through the conventional `host_vars` and `group_vars` variable files associated with an inventory. These variables provide host- or group-specific values used when Ansible runs automation. Red Hat documents inventory as the source for managed-node definitions and inventory variables, while the Ansible documentation describes host and group organization in detail. See Ansible inventory guide and Red Hat inventory documentation.

QUESTION NO: 6

State whether the following statement is true or false.

Ansible expects templates to use json format.

A. True

B. False

ANSWER: B

Explanation:

False is correct. Ansible templates are normally written using the Jinja templating language, not JSON. A template is a text file containing static content along with Jinja expressions, such as variables written as `{{ variable_name }}`, and control structures such as conditionals and loops. When an Ansible playbook uses the `ansible.builtin.template` module, Ansible evaluates the Jinja syntax and renders the result into a destination file on the managed host. The rendered output can be many kinds of text-based configuration or data formats, including INI-style service configuration, YAML, JSON, XML, or shell scripts. Therefore, JSON may be the desired *output* for a particular template, but it is not the format Ansible expects templates themselves to use. Jinja also provides filters such as `to_json` when a variable must be serialized as JSON within rendered content. Red Hat's Ansible documentation identifies templates as files processed through the Jinja template engine and documents the template module's handling of those files. See the Ansible template module documentation and the Ansible templating guide.

QUESTION NO: 7

What is the url for the official Ansible online documentation?

A. docs.ansible.com

B. help.ansible.com

C. google.com

D. ansiblehelp.com

ANSWER: A

Explanation:

The official online documentation for Ansible is available at docs.ansible.com. This is the primary documentation site maintained for Ansible and contains versioned documentation for Ansible Core, community packages, automation content such as collections, modules, plugins, playbooks, inventories, and command-line tooling. For exam preparation and real-world automation work, this site is the authoritative reference for module parameters, return values, examples, configuration settings, and supported behavior. In particular, administrators can use the Ansible Core documentation to verify how playbooks are structured, how variables and facts work, and how to use modules correctly. The documentation also provides direct navigation to content documentation and the latest maintained Ansible Core releases, making it the appropriate source when validating automation syntax or implementation details. Red Hat's Ansible Automation Platform documentation is separately hosted within the Red Hat customer portal, but the general official Ansible project documentation URL remains https://docs.ansible.com/.

QUESTION NO: 8

Which flags must be accepted as input for a dynamic inventory script?

A. Only --list

B. --host [hostname] and --list

C. --host [hostname] and --inv-list

D. --list and --format [file format]

ANSWER: B

Explanation:

The correct requirement is **--host [hostname] and --list**. Ansible invokes an inventory script with `--list` to obtain the complete inventory in JSON format. That output supplies the groups, their child groups, group variables, and host membership that Ansible uses to construct the inventory. The script must also support `--host HOSTNAME`, which returns JSON data for one requested host. This interface lets Ansible retrieve host-specific variables when needed, particularly for inventory scripts that do not provide all host variables in their full `--list` response.

The returned data must be valid JSON in Ansible's expected inventory structure. A script can be written in any executable language, but it must honor this command-line interface so that Ansible can query it consistently. Supporting both forms makes the script compatible with Ansible's inventory-script contract and permits both full inventory discovery and individual-host lookups. Red Hat's Ansible automation training and certification objectives commonly assess this standard dynamic-inventory behavior. See the Ansible documentation for developing dynamic inventory and the dynamic inventory guide.

QUESTION NO: 9

What is a typical use case for an Ansible Template?

A. Managing remote system configuration files

B. Defining a host group

C. Installing remote Ansible installations

D. Running remote commands

ANSWER: A

Explanation:

Managing remote system configuration files is a typical use case for an Ansible template. The `ansible.builtin.template` module processes a source template on the control node using the Jinja templating language, substitutes values from variables and facts, and then copies the rendered result to a destination path on a managed host. This is especially useful when a configuration file has a common structure but must contain host-specific or environment-specific values, such as a hostname, network address, port, service setting, or list of backend servers.

For example, an administrator can maintain one Jinja template for an application or web-server configuration and have Ansible render the appropriate final configuration for every managed system. Template files conventionally use the `.j2` extension, although it is not required. The task can also set ownership, permissions, and backup behavior for the deployed file, making templates a standard configuration-management mechanism in Ansible automation. See the official Ansible template module documentation and the Ansible playbook templating guide.

QUESTION NO: 10 - (SIMULATION)

Create a playbook /home/bob/ansible/timesync.yml that runs on hosts in the webservers host group and does the following:

• Uses the timesync RHEL system role.

• Sets the ntp server to 0.uk.pool.ntp.org

• Sets the timezone to UTC

ANSWER: See the explanation for the answer

Explanation:

Create `/home/bob/ansible/timesync.yml` with the following content:

```
---
- name: Configure time synchronization on web servers
  hosts: webservers
  become: true
  vars:
    timesync_ntp_servers:
      - hostname: 0.uk.pool.ntp.org
        iburst: true
    timesync_timezone: UTC
  roles:
    - timesync
```

Run it from the Ansible project directory:

```
cd /home/bob/ansible
ansible-playbook timesync.yml
```

The `timesync` RHEL system role configures the NTP source through `timesync_ntp_servers` and sets the system timezone through `timesync_timezone`.

QUESTION NO: 11 - (SIMULATION)

==========

control.realmX.example.com _ workstation.lab.example.com

node1.realmX.example.com _ servera.lab.example.com

node2.realmX.example.com _ serverb.lab.example.com

node3.realmX.example.com _ serverc.lab.example.com

node4.realmX.example.com _ serverd.lab.example.com

node5.realmX.example.com

- username:root, password:redhat

- username:admin, password:redhat

note1. don't change 'root' or 'admin' password.

note2. no need to create ssh-keygen for access, its pre-defined

note3. SELinux is in enforcing mode and firewalld is disabled/stop on whole managed hosts.

==========

Create and run an Ansible ad-hoc command.

--> As a system administrator, you will need to install software on the managed

nodes.

--> Create a shell script called yum-pack.sh that runs an Ansible ad-hoc command to

create yum-repository on each of the managed nodes as follows:

--> repository1

---

1. The name of the repository is EX407

2. The description is "Ex407 Description"

3. The base URL is http://content.example.com/rhel8.0/x86_64/dvd/BaseOS/

4. GPG signature checking is enabled

5. The GPG key URL is http://content.example.com/rhel8.0/x86_64/dvd/RPM-GPG-KEYredhat-

release

6. The repository is enabled

--> repository2

---

1. The name of the repository is EXX407

2. The description is "Exx407 Description"

3. The base URL is http://content.example.com/rhel8.0/x86_64/dvd/AppStream/

4. GPG signature checking is enabled

5. The GPG key URL is http://content.example.com/rhel8.0/x86_64/dvd/ RPM-GPG-KEYredhat-

release

6. The repository is enabled

ANSWER: See the explanation for the answer

Explanation:

Create the script in the Ansible project directory (for example, `/home/admin/ansible`). It uses the `yum_repository` ad-hoc module twice, once for each required repository.

```
cd /home/admin/ansible
cat > yum-pack.sh <<'EOF'
#!/bin/bash
ansible all -m ansible.builtin.yum_repository -a 'name=EX407 description="Ex407
Description" baseurl=http://content.example.com/rhel8.0/x86_64/dvd/BaseOS/ gpgcheck=yes
```

```
gpgkey=http://content.example.com/rhel8.0/x86_64/dvd/RPM-GPG-KEY-redhat-release enabled=yes
state=present'
ansible all -m ansible.builtin.yum_repository -a 'name=EXX407 description="Exx407
Description" baseurl=http://content.example.com/rhel8.0/x86_64/dvd/AppStream/ gpgcheck=yes
gpgkey=http://content.example.com/rhel8.0/x86_64/dvd/RPM-GPG-KEY-redhat-release enabled=yes
state=present'
EOF
chmod +x yum-pack.sh
./yum-pack.sh
```

This creates enabled repository definitions named `EX407` and `EXX407` on every host matched by the `all` inventory group. Both definitions enable GPG checking and set the specified GPG-key URL.

Optionally verify the result:

```
ansible all -m command -a 'dnf repolist all'
```

QUESTION NO: 12

By default, how many times can a role be applied during a play?

A. Once, no matter the parameters provided.

B. As many times as necessary, no matter the parameters.

C. Once, presuming it is only applied with a single set of parameters.

D. As many times as necessary, presuming the 'single_run' attribute is set to false.

ANSWER: C

Explanation:

Once, presuming it is only applied with a single set of parameters. is correct. Ansible performs role de-duplication within a play: when the same role is listed more than once with the same effective invocation, it runs only once. This behavior prevents the same role's tasks, handlers, variables, and dependencies from being needlessly processed repeatedly when a play includes the role through multiple paths or lists it redundantly.

A role can be invoked again during the same play when Ansible considers the invocation distinct, such as when it is supplied with different role parameters. The role author can also declare `allow_duplicates: true` in the role's `meta/main.yml` metadata to permit duplicate execution even with identical parameters. Therefore, the default rule is not an unconditional one-time limit; it is one execution for a given role invocation with a single parameter set. This distinction is especially important when roles are used as dependencies, because duplicate dependency entries are normally de-duplicated as well. See the Ansible documentation on role usage and de-duplication and role dependencies.

QUESTION NO: 13

Which are of the following are valid uses of a role's meta directory? (Choose all that apply.)

A. Setting variable values for use within a role.

B. Setting the allow_duplicates flag.

C. A role's dependency information.

D. Setting the become flag.

ANSWER: B C

Explanation:

A role's `meta` directory contains role metadata, conventionally stored in `meta/main.yml`. One supported use is declaring a role's dependency information. Dependencies are roles that Ansible processes before the role that declares them, allowing a role to express required prerequisite roles in a reusable, self-contained way. Each dependency can be named and can also receive variables, tags, and other role-invocation attributes as appropriate.

The metadata file also supports `allow_duplicates`. Setting this flag to `true` permits the role to run more than once when it is encountered through role dependency processing, including when equivalent role invocations would otherwise be de-duplicated. This is useful when repeated execution is intentional and each invocation needs to be honored. These metadata settings belong to the role itself, rather than to a play's variable declarations or privilege-escalation configuration, so they travel with the role when it is reused in other projects. See the Ansible documentation on role dependencies and the documented role duplication and execution behavior.

QUESTION NO: 14

Consider the following task:

```
- name: ensure application directory exists
  ansible.builtin.file:
    path: /srv/myapp/data
    state: directory
    owner: myapp
    group: myapp
    mode: '0750'
```

What does the `state: directory` setting request?

A. Ensure that the path exists as a directory.

B. Copy a directory from the control node.

C. Remove the directory and all of its contents.

D. Ensure that the path exists as an empty file.

ANSWER: A

Explanation:

The `state: directory` setting requests that Ansible ensure `/srv/myapp/data` exists as a directory. If the directory does not exist, the `ansible.builtin.file` module creates it. If it already exists, the module manages the requested ownership, group, permissions, and other supported file attributes without recreating it unnecessarily.

This behavior is idempotent and is commonly used to prepare application directories before deploying content or starting services. If intermediate directories in the specified path do not exist, Ansible creates them as necessary when creating the requested directory. See the ansible.builtin.file module documentation.

QUESTION NO: 15

Which keywords will cause a role to be applied in a playbook? (Choose all that apply.)

A. use_role

B. include_role

C. load_role

D. roles

E. import_role

ANSWER: B D E

Explanation:

The `roles` play keyword applies one or more roles to every host targeted by a play. Roles listed under `roles:` are processed as part of play execution, providing the role's variables, tasks, handlers, and other role content according to Ansible's role-loading behavior. The keyword is plural: `roles`, not `role`.

The `import_role` task keyword also applies a role. It performs a static import, meaning Ansible processes the imported role's task content while the playbook is parsed. This permits imported content to participate in parsing-time behavior, such as task listing and certain keyword inheritance.

The `include_role` task keyword applies a role dynamically at the point where that task executes. Because it is a dynamic inclusion, role execution can depend on runtime conditions, loops, and task flow. Together, `roles`, `import_role`, and `include_role` are the supported playbook mechanisms for applying roles. See the Ansible documentation on reusing roles and the import_role module documentation.

QUESTION NO: 16

Which of the following are valid default locations for Ansible configuration? (Choose all that apply.)

A. /etc/ansible/ansible.cfg

B. ${HOME}/.ansible.cfg

C. ${ANSIBLE_CONFIG}

D. /etc/ansible/config

ANSWER: A B C

Explanation:

Ansible obtains its configuration from an `ansible.cfg` file by searching a defined precedence list. The system-wide configuration file is `/etc/ansible/ansible.cfg`, which provides settings for all users when no higher-precedence configuration source is found. A user can instead maintain settings in `${HOME}/.ansible.cfg`; this per-user file is part of Ansible's standard configuration-file search order and takes precedence over the system-wide file. The `ANSIBLE_CONFIG` environment variable is also a valid configuration mechanism: when set, it specifies the path to the configuration file that Ansible must use and has the highest precedence among the standard file-location checks. This is useful when automation needs to select an explicit configuration file independently of the invoking user or system defaults. Ansible also checks for an `ansible.cfg` file in the current working directory, subject to its security handling for world-writable directories. The documented search order and environment-variable behavior are described in the Ansible configuration settings documentation and the configuration-file documentation.

QUESTION NO: 17

Which tags have special behavior when selecting tasks with the `ansible-playbook` command? (Choose all that apply.)

A. always

B. never

C. tagged

D. untagged

E. default

ANSWER: A B C D

Explanation:

The `always` and `never` tags have special selection behavior. Tasks tagged `always` run by default even when a run is limited with `--tags`, unless that tag is explicitly skipped. Tasks tagged `never` do not run by default; they run only when `never` or another tag applied to the same task is explicitly selected.

`tagged` and `untagged` are also special tag selectors recognized by the `--tags` option. `tagged` selects tasks that have at least one tag, while `untagged` selects tasks with no tags. `default` is not a special Ansible tag. See the Ansible playbook tags documentation.