DUMPSBOSS.

DevOps Tools Engineer

LPI 701-100

Version Demo

Total Demo Questions: 9

Total Premium Questions: 96

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	21
Topic 2, Container Management	32
Topic 3, Machine Deployment and Configuration	28
Topic 4, Service Operations	15
Total	96

QUESTION NO: 1

Which Ansible keyword is used to notify a handler when a task reports a change?

A. register

B. notify

C. listen

D. handler

E. changed_when

ANSWER: B

Explanation:

The correct keyword is `notify`. A task can include `notify` with the name of one or more handlers. When that task reports a changed state, Ansible queues the named handler to run, normally after the tasks in the current play have completed. This pattern is commonly used to restart or reload a service only when its configuration has actually changed.

For example, a template task that updates a web server configuration can use `notify: Restart web server`. The associated handler contains the service-management task. Because Ansible handlers are triggered only by changed tasks, unnecessary service restarts are avoided. See the Ansible handlers documentation.

QUESTION NO: 2

Which of the following statements is true about load balancers?

A. Load balancers are a security risk because they obfuscate the origin of connections.

B. Load balancer help to improve the availability and scalability of a service.

C. Load balancers are a single point of failure because they cannot be deployed redundantly.

D. Load balancer require access to private keys in order to be able to forward HTTPS traffic.

E. Load balancers cannot use connection content, such as HTTP cookies, to route traffic.

ANSWER: B

Explanation:

“Load balancer help to improve the availability and scalability of a service.” is correct. A load balancer distributes incoming requests or connections across multiple backend servers rather than directing all traffic to one instance. This distribution lets a service handle greater aggregate demand: additional backend instances can be added to the load-balancing pool as traffic grows, which is a core scalability pattern. It also improves availability because the load balancer can perform health checks and stop sending traffic to a backend that is unhealthy or unavailable, while continuing to route requests to healthy instances. In a properly designed highly available deployment, the load-balancing tier itself can also be redundant, allowing the service to continue operating when individual infrastructure components fail. Load balancing is therefore a common DevOps and cloud architecture mechanism for creating resilient, horizontally scalable services. The AWS Elastic Load Balancing documentation describes how load balancers distribute traffic across registered targets and route traffic only to healthy targets: AWS Elastic Load Balancing User Guide. The NGINX documentation likewise explains load balancing as a method for distributing client requests among application servers: NGINX HTTP Load Balancing.

QUESTION NO: 3

Which of the following HTTP methods are used by REST? (Choose three correct answers.)

A. CREATE

B. REPLACE

C. PUT

D. DELETE

E. GET

ANSWER: C D E

Explanation:

PUT, **DELETE**, and **GET** are standard HTTP request methods and are commonly used by RESTful APIs to operate on resources identified by URIs. GET retrieves a representation of a resource and is defined as a safe, read-only method. PUT creates or fully replaces the state of a resource at the target URI; its idempotent behavior makes it suitable when a client addresses a known resource location. DELETE requests removal of the resource identified by the target URI. REST is an architectural style that uses the uniform interface provided by HTTP, so REST APIs commonly map resource-oriented operations to HTTP's standardized semantics rather than using custom method names. The HTTP Semantics specification defines GET, PUT, and DELETE, including their intended behavior and properties. In practical REST API design, correctly using these methods allows clients, servers, intermediaries, caches, and monitoring tools to interpret requests consistently. See the HTTP GET method definition and the HTTP PUT and DELETE method definitions.

QUESTION NO: 4

Which of the following commands lists the nodes in a Docker Swam cluster?

A. docker-swarm listnodes

B. docker engine ls

C. docker node ls

D. docker machine ls

E. docker swarm nodes

ANSWER: C

Explanation:

The command `docker node ls` lists all nodes that belong to the current Docker Swarm. It displays each node's ID, hostname, status, availability, manager status, and engine version. This information lets an administrator quickly see which nodes are active, whether they can receive task assignments, and which nodes participate in the Swarm manager quorum. The command must be run from a Swarm manager node because managers maintain the cluster's authoritative state and membership information. It is commonly used when checking cluster health before deployments, maintenance operations, or node promotion and demotion. Docker documents `docker node ls` as the node-listing subcommand in the Docker node-management command group and notes the manager-node requirement. See the Docker CLI reference for docker node ls and the Docker Swarm administration guide.

QUESTION NO: 5

A Dockerfile contains the statements:

COPY data/ /data/

VOLUME /data

What happens when the resulting container is started without any additional volume configuration? (Choose two correct answers.)

A. Files existing in /data/ in the image are not available in the running container.

B. Changes to files within /data/ affect the Docker image and all other containers derived from it.

C. Existing files from /data/ in the image are copied to the new volume.

D. An error is raised because /data/ already contains data when the volume is mounted.

E. A new volume is created and mounted to /data/ within the new container.

ANSWER: C E

Explanation:

When an image declares `VOLUME /data`, Docker treats `/data` as a volume mount point. If the container is started without a user-supplied bind mount or named volume for that destination, Docker automatically creates an anonymous volume and mounts it at `/data` in that container. This gives the container writable storage that is separate from the image layer and persists independently of the container's writable layer.

Because the volume is empty when Docker creates it, Docker copies the files already present at `/data` in the image into that newly created volume by default. Therefore, the content added by `COPY data/ /data/` is available to the started container through the mounted volume. Subsequent modifications are made to the volume's data, rather than modifying the immutable image. Docker documents this behavior as volume population: when an empty volume is mounted into a directory that already contains files in the container, those files are propagated into the volume. See the Docker volumes documentation and the Dockerfile VOLUME reference.

QUESTION NO: 6

How is cloud-init integrated with a managed system image?

A. cloud-init provides the cloud-init-worker command which has to be invoked periodically within the running instance.

B. cloud-init provides systemd units which must be included in several stages of the booting process of the instance.

C. cloud-init provides its own startup mechanism which replaces the instance's original init system, such as systemd.

D. cloud-init provides a Linux kernel module that must be included and loaded in the instance's initramfs.

E. cloud-init provides the cloud-init-daemon service which is launched during startup and keeps the instance in sync with the desired configuration.

ANSWER: B

Explanation:

cloud-init provides systemd units which must be included in several stages of the booting process of the instance. In a systemd-based managed image, cloud-init integrates with the operating system's normal init and service-management framework rather than replacing it. Its package installs several units that run at defined points in startup so that initialization can occur in the required order. Typical stages include early local initialization, network-dependent initialization, configuration processing, and final processing. This staged design lets cloud-init discover its datasource, establish any required networking, retrieve instance metadata and user data, apply configuration such as users, SSH keys, packages, and files, and then execute final modules only after their prerequisites are available.

The relevant units include `cloud-init-local.service`, `cloud-init.service`, `cloud-config.service`, and `cloud-final.service`. Their ordering and dependencies allow an image to be configured predictably on first boot or when a new instance is created from that image. Cloud-init's official boot-stage documentation describes these stages and their relationship to system startup. See the cloud-init boot stages documentation and the cloud-init systemd integration documentation.

QUESTION NO: 7

Which of the following HTTP methods are idempotent according to HTTP semantics? (Choose three correct answers.)

A. GET

B. POST

C. PUT

D. DELETE

E. CONNECT

ANSWER: A C D

Explanation:

An idempotent HTTP method has the same intended effect on the server when a request is made once or multiple times. `GET` is safe and idempotent because it requests a representation without requesting a state change. `PUT` is idempotent because repeated requests with the same representation request the same replacement state for the target resource. `DELETE` is also idempotent: after a resource has been deleted, later identical deletion requests do not request an additional state change, even though their response status may differ.

`POST` is not generally idempotent because repeating a POST can create multiple subordinate resources or otherwise perform an action more than once. HTTP semantics are defined in RFC 9110. See the HTTP idempotent methods definition.

QUESTION NO: 8

Which of the following tasks are completed by docker-compose down when it is used with additional parameters? (Choose two correct answers.)

A. Delete all volumes defined in the Compose file.

B. Delete all containers defined in the composer file.

C. Delete all networks defined in the composer file.

D. Delete all images used by the services in the Compose file from the local Docker host.

E. Delete all images built from the composer file from their registry.

ANSWER: A D

Explanation:

The `docker-compose down` command supports options that extend its normal teardown behavior. Supplying `--volumes` (or `-v`) removes the named volumes declared in the Compose file as well as anonymous volumes attached to containers. This is useful when an environment must be fully reset, including persistent application data. Docker documents that external volumes are not removed, because Compose does not manage their lifecycle.

Supplying `--rmi all` removes images used by the services in the Compose application. The related `--rmi local` form removes only images that do not have a custom tag. Image removal occurs on the Docker host where Compose is executed; Compose does not delete images from a remote registry. These parameters allow a teardown workflow to clean up storage resources and locally cached service images in addition to the standard Compose project resources.

See the Docker CLI reference for docker compose down and Docker's guide to Compose file behavior.

QUESTION NO: 9 - (SIMULATION)

Which docker subcommand starts a new container? (Specify only the subcommand without any path or parameters.)

ANSWER: See the explanation for the answer

Explanation:

`run`

The `docker run` command creates and starts a new container from an image.