DUMPSBOSS.

# LPIC-3: Virtualization and Containerization

## LPI 305-300

### Version Demo

**Total Demo Questions: 20**

**Total Premium Questions: 202**

**Buy Premium PDF**

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, Virtualization | 137 |
| Topic 2, Containerization | 55 |
| Topic 3, Mix Questions | 10 |
| Total | 202 |

QUESTION NO: 1

**What is Packer?**

**A.** A container orchestration tool

**B.** An image building tool

**C.** A configuration management tool

**D.** A virtualization platform

**ANSWER: B**

**Explanation:**

Packer is an image-building tool from HashiCorp that automates the creation of machine images. It takes a reusable template as input, provisions and configures an operating-system environment, and produces one or more identical images for target platforms. A single Packer template can support several builders, allowing an organization to generate consistent images for environments such as VMware, VirtualBox, public-cloud services, and other supported platforms from the same definition.

This role is valuable in virtualized and cloud environments because machine images provide a standardized, preconfigured starting point for deployments. By defining image construction as code, teams can version, review, test, and repeatedly rebuild images, reducing configuration drift and making releases more reproducible. Packer can also use provisioners during the build process to install packages, apply system configuration, or run scripts before the final image is created. HashiCorp describes Packer as a tool for creating identical machine images for multiple platforms from a single source configuration. See the official Packer documentation and the Packer introduction.

**QUESTION NO: 2**

What is the purpose of capabilities in the context of container virtualization?

**A.** Map potentially dangerous system calls to an emulation layer provided by the container virtualization.

**B.** Restrict the disk space a container can consume.

**C.** Enable memory deduplication to cache files which exist in multiple containers.

**D.** Allow regular users to start containers with elevated permissions.

**E.** Prevent processes from performing actions which might infringe the container.

**ANSWER: E**

**Explanation:**

“Prevent processes from performing actions which might infringe the container.” is correct because Linux capabilities divide traditional root privileges into distinct, independently controllable permissions. Rather than granting a container process unrestricted superuser authority, a container runtime can give it only the capabilities required for its intended workload and remove capabilities that are unnecessary or security-sensitive. This supports the principle of least privilege while retaining necessary functionality within the container. For example, permissions relating to mounting filesystems, changing network configuration, loading kernel modules, altering the system clock, or bypassing discretionary access controls can be withheld. Importantly, being root inside a container does not need to mean possessing every privileged operation on the host: the process is constrained by its capability sets as well as by other isolation mechanisms. Container engines commonly apply a restricted default capability set and provide configuration controls to explicitly add or drop individual capabilities for a workload. This limits the impact of a compromised process and helps preserve isolation between the container and the host environment. The Linux kernel documents the individual capability permissions in capabilities(7), and Docker documents capability management for containers in its runtime privilege and Linux capabilities guide.

**QUESTION NO: 3**

What is true about containerd?

**A.** It is a text file format defining the build process of containers.

**B.** It runs in each Docker container and provides DHCP client functionality

**C.** It uses runc to start containers on a container host.

**D.** It is the initial process run at the start of any Docker container.

**E.** It requires the Docker engine and Docker CLI to be installed.

**ANSWER: C**

**Explanation:**

containerd uses `runc` to start containers on a container host. containerd is a high-level container runtime that manages the complete lifecycle of containers, including image transfer and storage, snapshot management, container creation, task execution, and lifecycle events. It was originally extracted from Docker and is now a graduated CNCF project used directly by platforms such as Kubernetes through its Container Runtime Interface integration.

When a container process must be created, containerd delegates the low-level runtime work to an OCI-compliant runtime. The standard implementation commonly used for this purpose is `runc`. runc creates and runs containers according to the Open Container Initiative Runtime Specification, including configuring Linux namespaces, cgroups, mounts, and the container process. containerd normally uses a shim process to supervise the running container and maintain its lifecycle independently of the containerd daemon. The term in the statement must therefore be `runc`, not “rune.” This relationship between containerd as the lifecycle manager and runc as the OCI runtime is central to the container runtime architecture tested in LPIC-3 virtualization and containerization objectives.

References: containerd project; Open Container Initiative runc runtime.

**QUESTION NO: 4**

The command virsh vol-list vms returns the following error:

error: failed to get pool 'vms'

error: Storage pool not found: no storage pool with matching name 'vms '

Given that the directory /vms exists, which of the following commands resolves this issue?

**A.** dd if=/dev/zero of=/vms bs=1 count=0 flags=name:vms

**B.** libvirt-poolctl new ---name=/vms ---type=dir ---path=/vms

**C.** qemu-img pool vms:/vms

**D.** virsh pool-create-as vms dir --target /vms

**E.** touch /vms/.libvirtpool

**ANSWER: D**

**Explanation:**

`virsh pool-create-as vms dir --target /vms` is correct because it creates a libvirt storage-pool definition named `vms`, using the directory `/vms` as the pool target. Libvirt storage pools are managed objects with a name, type, and target path; the existence of a directory in the host filesystem does not by itself make that directory available as a pool to `virsh`. The reported error specifically concerns lookup of a pool by its libvirt name, so creating the named directory-type pool supplies the missing object that `virsh vol-list vms` needs to query.

The `pool-create-as` command creates and starts the pool immediately, which resolves the immediate command failure. A directory pool is appropriate when virtual-machine disk images or other volumes are stored as ordinary files below a host

directory. For an environment where the pool must survive a libvirt service restart or host reboot, the corresponding persistent approach is to define the pool, for example with `virsh pool-define-as`, then start it and configure autostart as needed. Libvirt documents both the storage-pool concepts and the `virsh pool-create-as` command syntax at libvirt Storage Management and virsh pool-create-as documentation.

## QUESTION NO: 5

**What are the key benefits of using cloud management tools?**

**A.** Streamlined billing and invoicing

**B.** Automated scaling of virtual machines

**C.** Efficient resource allocation

**D.** Enhanced security through encryption

## ANSWER: C

**Explanation:**

Efficient resource allocation is a fundamental benefit of cloud management tools because these platforms provide centralized control over compute, storage, and networking capacity across cloud infrastructure. Administrators can use policies, quotas, scheduling, monitoring, and orchestration to place workloads on suitable resources and to balance capacity as demand changes. This reduces idle capacity while helping ensure that virtual machines, containers, and their associated services receive the resources needed to meet operational requirements.

In practical cloud environments, resource allocation is not merely a one-time provisioning task. Cloud management continuously tracks available and consumed capacity, applies tenant or project limits, and supports automated placement decisions. These capabilities improve utilization, support scalable operations, and give organizations a consistent way to manage resources across private, public, or hybrid environments. The OpenStack Placement service, for example, tracks inventories and allocations of cloud resources so that schedulers can make placement decisions based on available capacity. This illustrates why allocating resources efficiently is central to cloud management rather than an incidental feature.

The LPI objectives for LPIC-3 Virtualization and Containerization include cloud-management concepts, while OpenStack's documentation describes the resource inventory and allocation model used by a cloud-management platform. See the LPI 305 exam objectives and the OpenStack Placement documentation.

## QUESTION NO: 6

Which of the following commands can be used to determine whether the local machine is capable of running full virtualized Xen guests? (Choose TWO correct answers.)

**A.** dmesg |grep -i intel|grep -i vt; dmesg |grep -i amd|grep -i v

**B.** egrep '(vmx|svm)' /proc/cpuinfo

**C.** xl dmesg | grep -i hvm

**D.** grep -i "Full Virtualization" /proc/xen

**E.** grep -i "Full Virtualization" /etc/xen/*

## ANSWER: B C

**Explanation:**

Fully virtualized Xen guests, commonly called HVM guests, require processor hardware-virtualization support. On Intel processors this capability is exposed through the `vmx` CPU feature flag, while AMD processors expose it through `svm`. Running `egrep '(vmx|svm)' /proc/cpuinfo` checks the feature flags exported by the running Linux kernel and is

therefore a direct way to establish whether the processor offers the necessary virtualization extension. This check is useful before Xen has been installed or booted.

When Xen is already the active hypervisor, `xl dmesg | grep -i hvm` queries Xen's own hypervisor log and filters it for HVM-related initialization messages. Those messages report Xen's detection and setup of hardware-assisted virtualization support, providing confirmation from the component that will run the guests. The `xl dmesg` command is documented as displaying the Xen hypervisor message buffer, making it suitable for checking boot-time capability information on an operational Xen host. Xen's HVM documentation also describes HVM as hardware-assisted virtualization based on Intel VT-x or AMD-V. See the xl dmesg manual page and the Xen Project HVM documentation.

## QUESTION NO: 7

Which of the following statements are true regarding a load balancer? (Choose TWO correct answers.)

**A.** Failover clusters are not operational without at least one dedicated load balancer.

**B.** All response packages sent from a backend server to a client must pass through the load balancer.

**C.** The load balancer is usually deployed on one of the backend servers.

**D.** Load balancers may be able to analyze traffic.

**E.** All incoming connections to the cluster services are received by the load balancer.

## ANSWER: D E

**Explanation:**

"Load balancers may be able to analyze traffic" is correct because load balancing can operate at more than the network and transport layers. Layer 7 load balancers and reverse proxies can inspect application-protocol information, particularly HTTP host headers, paths, methods, cookies, and sometimes request content. This enables content-based routing, such as sending requests for a particular URL path to a specific backend pool. Depending on the implementation, the load balancer can also terminate TLS, perform health checks, maintain session persistence, and apply application-aware policies before forwarding a request.

"All incoming connections to the cluster services are received by the load balancer" correctly describes the normal client-facing load-balancer architecture. Clients connect to a virtual IP address or virtual service exposed by the load balancer rather than selecting an individual backend node. The load balancer accepts the connection, evaluates backend availability through health checks, and selects an eligible server according to its configured scheduling policy. Presenting this single entry point allows backend servers to be added, removed, or taken out of service while preserving a stable service address for clients. See the NGINX load-balancing overview and the HAProxy load-balancing documentation.

## QUESTION NO: 8

Which of the following tasks are part of a hypervisorâ€™s responsibility? (Choose two.)

**A.** Create filesystems during the installation of new virtual machine quest operating systems.

**B.** Provide host-wide unique PIDs to the processes running inside the virtual machines in order to ease inter-process communication between virtual machines.

**C.** Map the resources of virtual machines to the resources of the host system.

**D.** Manage authentication to network services running inside a virtual machine.

**E.** Isolate the virtual machines and prevent unauthorized access to resources of other virtual machines.

**F.** Map the resources of virtual machines to the resources of the host system, and isolate the virtual machines to prevent unauthorized access to resources of other virtual machines.

**ANSWER: F**

**Explanation:**

“Map the resources of virtual machines to the resources of the host system, and isolate the virtual machines to prevent unauthorized access to resources of other virtual machines” is correct because these are core hypervisor functions. A hypervisor virtualizes underlying physical resources, including processor time, memory, storage, and network interfaces, and presents controlled virtual hardware to each guest. It schedules and mediates guest access to those physical resources so multiple virtual machines can execute concurrently on one host without directly controlling the host hardware. The hypervisor also establishes isolation boundaries between guests. Each virtual machine operates in its own execution environment, with its own virtual devices and address spaces, and access to resources assigned to another guest is controlled rather than implicitly shared. This isolation is fundamental to multi-tenant security, workload stability, and fault containment. The LPI LPIC-3 Virtualization and Containerization certification covers virtualization concepts and hypervisor-based workload operation; see the LPIC-3 305 overview. Red Hat’s virtualization documentation also describes the hypervisor as the layer that abstracts host resources and runs isolated virtual machines: Introduction to virtualization. Because the supplied item is designated Single Choice while its stem requests two selections, the combined statement is required to express the complete correct answer as one selectable option.

## QUESTION NO: 9

What is the default provider of Vagrant?

**A.** lxc

**B.** hyperv

**C.** virtualbox

**D.** vmware_workstation

**E.** docker

**ANSWER: C**

**Explanation:**

**virtualbox** is the default provider used by Vagrant when no provider is explicitly selected. A Vagrant provider is the virtualization or container platform responsible for creating and managing the environment defined by a Vagrantfile. Vagrant supports several providers, but VirtualBox has historically been the built-in default because it is freely available, works across major host operating systems, and provides a straightforward starting point for local development environments. Consequently, running commands such as `vagrant up` without a `--provider` argument causes Vagrant to use VirtualBox when it is installed and available. The selected provider can be overridden for an individual command with `vagrant up --provider=...`. It can also be changed for a shell environment by setting the `VAGRANT_DEFAULT_PROVIDER` environment variable. Vagrant’s provider documentation identifies VirtualBox as the default provider and describes how Vagrant detects and uses installed providers. See the Vagrant provider documentation and the VirtualBox provider documentation for provider behavior, installation, and configuration details.

## QUESTION NO: 10

Which of the following statements in aDockerfileleads to a container which outputs hello world? (Choose two.)

**A.** ENTRYPOINT "echo hello world"

**B.** ENTRYPOINT [ "echo hello world" ]

**C.** ENTRYPOINT [ "echo", "hello", "world" ]

**D.** ENTRYPOINT echo hello world

**E.** ENTRYPOINT "echo", "Hello", "World*

**ANSWER: A C D**

**Explanation:**

`ENTRYPOINT "echo hello world"` and `ENTRYPOINT echo hello world` use the shell form of the Dockerfile `ENTRYPOINT` instruction. In shell form, Docker runs the instruction through the image's default shell, conventionally as `/bin/sh -c` on Linux images. Consequently, the shell parses and executes `echo hello world` when a container starts, producing the requested output. Quoting the complete shell command does not prevent it from being interpreted by that shell.

`ENTRYPOINT [ "echo", "hello", "world" ]` is also valid and produces the same output. This is the exec form, represented by a JSON array. Its first element identifies the executable, `echo`, and the remaining elements are passed as its arguments. This form directly starts the specified executable rather than adding a shell, and it is generally preferred when predictable argument handling and signal delivery to the container's main process are important.

Docker permits both shell-form and exec-form `ENTRYPOINT` instructions. The key exec-form requirement is that the executable and each argument occupy separate JSON-array elements. See the Dockerfile ENTRYPOINT reference and Docker's Dockerfile best-practices guidance.

**QUESTION NO: 11**

Which of the following resources are directly (i.e. without specialized machine images) available to users of an OpenStack cloud? (Choose THREE correct answers.)

**A.** Database servers to store relational data.

**B.** Virtual machines for computing tasks.

**C.** Application servers for immediate deployment with Mono applications.

**D.** Object storage that is accessed through an API.

**E.** Block storage devices for persistent data storage.

**ANSWER: B D E**

**Explanation:**

OpenStack directly provides infrastructure-as-a-service resources through its tenant-facing APIs and dashboard. "Virtual machines for computing tasks" are supplied by the Nova compute service: a tenant selects an available base image, flavor, network, and related resources to create an instance. This is a general-purpose compute capability rather than a specialized application appliance. See the OpenStack Nova documentation.

"Object storage that is accessed through an API" is a native OpenStack resource provided by Swift. Swift stores data as objects in containers and exposes HTTP-based APIs for creating, retrieving, and managing that data, making it available independently of a guest operating-system image. The OpenStack Swift documentation describes this object-storage service and its API-oriented model.

"Block storage devices for persistent data storage" are supplied by Cinder. Cinder lets tenants create persistent volumes, manage their lifecycle, and attach them to compute instances when required. The volume exists as a cloud storage resource in its own right, rather than being an application-specific server image. Together, compute instances, object storage, and persistent block volumes are core OpenStack infrastructure resources directly consumable by cloud users.

**QUESTION NO: 12**

Which of the following mechanisms are used by LXC and Docker to create containers? (Choose three.)

**A.** Linux Capabilities

**B.** Kernel Namespaces

**C.** Control Groups

**D.** POSIXACLs

**E.** File System Permissions

**ANSWER: A B C**

**Explanation:**

LXC and Docker create containers by combining Linux kernel features that isolate processes and regulate their access to host resources. **Kernel Namespaces** establish distinct views of system-wide resources. For example, PID namespaces give a container its own process-numbering tree, network namespaces provide separate network interfaces and routing, and mount namespaces isolate filesystem mount points. Other namespace types isolate hostnames, IPC resources, and user/group IDs.

**Control Groups**, commonly called cgroups, organize processes into groups to account for and control resource consumption. Container runtimes use them to apply CPU, memory, I/O, and process-count limits, enabling multiple workloads to share a host with defined resource boundaries.

**Linux Capabilities** divide traditional root authority into individually assignable privileges. LXC and Docker use capabilities to grant only the privileges a containerized process requires and to remove unnecessary high-impact privileges. This supports the reduced-privilege execution model used alongside namespace isolation and cgroup resource governance. These mechanisms are fundamental Linux container primitives, with the runtime configuring them when a container is started. See the Linux kernel documentation on namespaces and Docker's overview of container security.

## QUESTION NO: 13

Which functionality is provided by Vagrant as well as by Docker? (Choose three.)

**A.** Both can share directories from the host file system to a guest.

**B.** Both start system images as containers instead of virtual machines by default.

**C.** Both can download required base images.

**D.** Both can apply changes to a base image.

**E.** Both start system images as virtual machines instead of containers bv default.

**ANSWER: A C D**

**Explanation:**

"Both can share directories from the host file system to a guest" is correct because each tool supports making host-side files available inside its managed environment. Vagrant defines synced folders in a Vagrantfile, while Docker supports bind mounts and volumes when containers are run. This supports common development workflows in which source code remains on the host but is used by the VM or container. "Both can download required base images" is also correct: Vagrant obtains versioned box images from configured box sources, and Docker retrieves image layers from registries. These reusable starting environments make it possible to create consistent development or deployment environments without manually installing an operating system and dependencies each time. Finally, "Both can apply changes to a base image" is correct in the sense that both provide mechanisms to customize a starting environment. Vagrant can provision a machine created from a box and package that configured machine as a reusable box; Docker builds a new image from a parent image using Dockerfile instructions. Thus, both support sharing host data, obtaining reusable base environments, and producing customized environments based on those foundations. See the Vagrant box documentation and Docker's Dockerfile documentation.

## QUESTION NO: 14

Which of the following statements are true of full virtualization? (Select THREE correct answers)

**A.** Full virtualization has superior I/O performance through the emulated device drivers.

**B.** Full virtualization is faster than paravirtualization.

**C.** Full virtualization requires time and system resources for emulation.

**D.** Full virtualization requires no modification to the Guest OS kernel.

**E.** Full virtualization works through CPU emulation.

**ANSWER: C D E**

**Explanation:**

Full virtualization presents each guest with a complete virtual hardware platform, including virtual processors, memory, storage controllers, network interfaces, firmware, and other devices. Because the hypervisor supplies an environment that behaves like physical hardware, an operating system can run without changes to its kernel. This is the defining practical benefit expressed by “Full virtualization requires no modification to the Guest OS kernel.”

Providing that hardware-compatible environment requires the hypervisor to handle privileged processor activity and device access. Depending on the platform and workload, this involves instruction emulation, hardware-assisted virtualization, traps or VM exits, memory-translation support, and virtual-device emulation. These activities consume processing time and system resources, making “Full virtualization requires time and system resources for emulation” accurate.

“Full virtualization works through CPU emulation” correctly identifies the CPU-virtualization mechanism central to a fully virtual machine. On modern systems this is commonly accelerated by processor features such as Intel VT-x or AMD-V rather than performed exclusively through software interpretation, but the hypervisor must still virtualize the CPU execution environment and privileged operations. QEMU distinguishes full system emulation from virtualization acceleration in its system emulation and virtualization introduction. Red Hat also describes how a hypervisor creates and manages virtual machines in its overview of virtualization.

**QUESTION NO: 15**

What is the purpose of the kubelet service in Kubernetes?

**A.** Provide a command line interface to manage Kubernetes.

**B.** Build a container image as specified in a Dockerfile.

**C.** Manage permissions of users when interacting with the Kubernetes API.

**D.** Run containers on the worker nodes according to the Kubernetes configuration.

**E.** Store and replicate Kubernetes configuration data.

**ANSWER: D**

**Explanation:**

Run containers on the worker nodes according to the Kubernetes configuration. This describes the kubelet’s core purpose: it is the primary node agent that runs on each Kubernetes node. The kubelet receives Pod specifications for workloads scheduled to its node and continuously works to ensure that the containers specified in those Pods are running as intended. To do this, it communicates with the configured container runtime through the Container Runtime Interface, manages container lifecycle operations, and helps prepare required local resources such as mounted volumes. It also observes the state of Pods and the node, then reports relevant status information back to the Kubernetes control plane. The kubelet does not independently decide where a Pod should run; scheduling is a control-plane responsibility. Instead, after a Pod has been assigned to a node, the kubelet is responsible for making that desired workload state real on that node and monitoring it over time. Kubernetes documents the kubelet as an agent that runs on every node and ensures containers described in PodSpecs are running and healthy. See the Kubernetes Components overview and the kubelet reference documentation.

**QUESTION NO: 16**

Which one of the following options represents a valid declaration of a backend server with HAProxy?

**A.** host server_x 10.1.0.1:80 cookie server_x

**B.** backend server_x 10.1.0.1:80

**C.** server server_x 10.1.0.1:80 check

**D.** pool server_x 10.1.0.1:80

**ANSWER: C**

**Explanation:**

`server server_x 10.1.0.1:80 check` is a valid HAProxy backend-server declaration. HAProxy defines individual servers using the `server` directive within a `backend` or `listen` section. Its required structure is `server <name> <address>[:port]`, followed by any optional server parameters. Here, `server_x` is the locally unique server name and `10.1.0.1:80` supplies the server IPv4 address and TCP port. The optional `check` keyword enables active health checks, allowing HAProxy to periodically test the server's availability and to avoid forwarding new traffic to it when checks determine that it is unavailable. A typical complete configuration would place this declaration below a section header such as `backend application_servers`. Server declarations can also include settings such as weights, TLS options, connection limits, or cookie values, but none of these are required for this basic valid form. The HAProxy configuration reference documents `server` as the directive for declaring a backend server and specifies the address-and-port form used here. See the HAProxy Configuration Manual and the HAProxy Configuration Manual documentation.

**QUESTION NO: 17**

Which of the following resources can be limited by libvirt for a KVM domain? (Choose two.)

**A.** Amount of CPU time

**B.** Size of available memory

**C.** File systems allowed in the domain

**D.** Number of running processes

**E.** Number of available files

**ANSWER: A B**

**Explanation:**

**Amount of CPU time** is a resource that libvirt can control for a KVM domain through its domain resource-tuning configuration. Libvirt exposes CPU scheduling controls that integrate with the Linux kernel’s control-group facilities. For example, a domain can receive relative CPU shares, be assigned a quota and period that cap its CPU usage, or have virtual CPUs pinned to specific host CPUs. These controls allow an administrator to prevent one virtual machine from monopolizing host processor capacity while allocating CPU time according to workload priority and policy.

**Size of available memory** can likewise be constrained in the domain XML definition. Libvirt supports specifying memory values such as the domain’s maximum memory and the memory currently assigned to the guest. Depending on the guest configuration and available virtualization features, memory ballooning can be used to adjust the guest’s active memory allocation within the configured limits. These settings let administrators plan host memory overcommit and ensure that individual KVM guests remain within an intended memory budget.

Libvirt documents these capabilities in its domain XML resource-tuning and memory-allocation sections. See the libvirt CPU tuning documentation and the libvirt memory allocation documentation.

**QUESTION NO: 18**

Which of the following statements are true regarding a Pod in Kubernetes? (Choose two.)

**A.** All containers of a Pod run on the same node.

**B.** Pods are always created automatically and cannot be explicitly configured.

**C.** A Pod is the smallest unit of workload Kubernetes can run.

**D.** When a Pod fails, Kubernetes restarts the Pod on another node by default.

**E.** systemd is used to manage individual Pods on the Kubernetes nodes.

**ANSWER: A C**

**Explanation:**

All containers of a Pod run on the same node because a Pod is Kubernetes' atomic scheduling unit. The scheduler selects a single node for the Pod, and the kubelet on that node runs the Pod's containers. This co-location is fundamental to the Pod model: containers in the same Pod can communicate through the shared Pod network namespace, including the same IP address and port space, and can mount shared volumes. This arrangement supports tightly coupled processes, such as an application container and a helper or proxy container that must operate together.

A Pod is the smallest unit of workload Kubernetes can run because Kubernetes deploys and schedules Pods rather than scheduling individual containers independently. A Pod may contain one container, which is the common case, or multiple closely related containers. Higher-level workload resources such as Deployments, StatefulSets, DaemonSets, and Jobs define desired workload behavior and create or manage Pods to implement that behavior. Consequently, Pods are the direct runnable workload objects that Kubernetes places onto nodes and whose lifecycle is managed by the cluster.

See the Kubernetes Pods documentation and the Kubernetes scheduler documentation.

## QUESTION NO: 19 - (SIMULATION)

Which command in the KVM monitor restores a snapshot?

**ANSWER: See the explanation for the answer**

**Explanation:**

The QEMU/KVM monitor command to restore a named snapshot is:

```
loadvm SNAPSHOT_NAME
```

For example:

```
(qemu) loadvm before-upgrade
```

`loadvm` reverts the virtual machine to a snapshot previously created with `savevm`.

## QUESTION NO: 20

Which command displays the status of Docker Swarm nodes from a Swarm manager?

**A.** docker node ls

**B.** docker swarm ps

**C.** docker cluster ls

**D.** docker service nodes

**E.** docker swarm status

**ANSWER: A**

**Explanation:**

The correct command is `docker node ls`. When executed on a Swarm manager, it lists nodes known to the Swarm and displays information including node ID, hostname, availability, manager status, and engine version. It can be used to confirm which nodes are active managers and workers and whether a node is available for task scheduling.

This command is a manager-side operation because managers maintain the cluster state. A worker node runs assigned tasks but does not have the cluster-management information needed to list all Swarm nodes. See the docker node ls command reference and the Docker Swarm nodes documentation.