Question 1

Question Type: MultipleChoice

You are running only Kubernetes workloads on a worker node that requires

maintenance, such as installing patches or an OS upgrade.

Which command must be run on the node to gracefully terminate all pods on

the node, while marking the node as unschedulable?

Options:

A- `docker swarm leave'

B- `docker node update -availability drain <node name>

C- `kubectl drain <node name>'

D- `kubectl cordon <node name>

Answer:

C

Explanation:

The command kubectl drain <node name> is the correct one to run on the node to gracefully terminate all pods on the node, while marking the node as unschedulable. This command will safely evict all the pods from the node before you perform maintenance on the node, such as installing patches or an OS upgrade1. It will respect the PodDisruptionBudgets you have specified, if any, and allow the pod's containers to gracefully terminate1. It will also mark the node as unschedulable, so that no new pods can be scheduled on the node until it is ready1.

The other commands are not correct because:

* docker swarm leave will make the node leave the swarm cluster, but it will not affect the Kubernetes workloads on the node2.

* docker node update -availability drain <node name> will change the availability of the node to drain, which means that no new tasks can be assigned to the node, but it will not terminate the existing pods on the node3.

* kubectl cordon <node name> will mark the node as unschedulable, but it will not evict the pods on the node4.

* Safely Drain a Node | Kubernetes

* [docker swarm leave | Docker Docs]

* [docker node update | Docker Docs]

* [kubectl cordon | Kubernetes Docs]

Question 2

Question Type: **MultipleChoice**

Which docker run` flag lifts cgroup limitations?

Options:

A- `docker run -privileged

B- `docker run -cpu-period

C- `docker run -isolation

D- `docker run -cap-drop

Answer:

A

Explanation:

The --privileged flag lifts all the cgroup limitations for a container, as well as other security restrictions imposed by the Docker daemon1. This gives the container full access to the host's devices, resources, and capabilities, as if it was running directly on the host2. This can be useful for certain use cases that require elevated privileges, such as running Docker-in-Docker or debugging system issues3.

However, using the --privileged flag also poses a security risk, as it exposes the host to potential attacks or damages from the container4. Therefore, it is not recommended to use the --privileged flag unless absolutely necessary, and only with trusted images and containers.

The other options are not correct because they do not lift all the cgroup limitations for a container, but only affect specific aspects of the container's resource allocation or isolation:

* The --cpu-period flag sets the CPU CFS (Completely Fair Scheduler) period for a container, which is the length of a CPU cycle in microseconds. This flag can be used in conjunction with the --cpu-quota flag to limit the CPU time allocated to a container. However, this flag does not affect other cgroup limitations, such as memory, disk, or network.

* The --isolation flag sets the isolation technology for a container, which is the mechanism that separates the container from the host or other containers. This flag is only available on Windows containers, and can be used to choose between process, hyperv, or process-isolated modes. However, this flag does not affect the cgroup limitations for a container, but only the level of isolation from the host or other containers.

* The --cap-drop flag drops one or more Linux capabilities for a container, which are the privileges that a process can use to perform certain actions on the system. This flag can be used to reduce the attack surface of a container by removing unnecessary or dangerous capabilities. However, this flag does not affect the cgroup limitations for a container, but only the capabilities granted to the container by the Docker daemon.

* Runtime privilege and Linux capabilities

* Docker Security: Using Containers Safely in Production

* Docker run reference

* Docker Security: Are Your Containers Tightly Secured to the Ship? SlideShare

* [Secure Engine]

* [Configure a Pod to Use a Limited Amount of CPU]

* [Limit a container's resources]

* [Managing Container Resources]

* [Isolation modes]

* [Windows Container Isolation Modes]

* [Windows Container Version Compatibility]

* [Docker and Linux Containers]

* [Docker Security Cheat Sheet]

* [Docker Security: Using Containers Safely in Production]

Question 3

Question Type: MultipleChoice

You are pulling images from a Docker Trusted Registry installation

configured to use self-signed certificates, and this error appears:

`x509: certificate signed by unknown authority.

You already downloaded the Docker Trusted Registry certificate authority

certificate from https://dtr.example.com/ca.

How do you trust it? (Select two.)

Options:

A- Pass '-trust-certificate ca.crt to the Docker client.

B- Place the certificate in '/etc/docker/dtr/dtr.example.com.crt' and restart the Docker daemon on all cluster nodes.

C- Place the certificate in /etc/docker/certs.d/dtr.example.com/ca.crt' on all cluster nodes.

D- Pass -- insecure-registry to the Docker client.

E- Place the certificate in your OS certificate path, trust the certificate system-wide, and restart the Docker daemon across all cluster nodes.

Answer:

C, E

Explanation:

To trust a self-signed certificate from a Docker Trusted Registry (DTR), you need to place the certificate in the appropriate location on all cluster nodes and restart the Docker daemon. There are two possible locations for the certificate, depending on your OS and Docker version1:

* /etc/docker/certs.d/dtr.example.com/ca.crt: This is the preferred location for Linux systems and Docker versions 1.13 and higher. This directory is scanned by Docker for certificates and keys for each registry domain2.

* Your OS certificate path: This is the fallback location for other OSes and Docker versions. You need to find the certificate store for your OS and copy the certificate there. You also need to trust the certificate system-wide, which may require additional steps depending on your OS3.

The other options are not correct because:

* Passing '-trust-certificate ca.crt to the Docker client is not a valid option. There is no such flag for the Docker client4.

* Placing the certificate in '/etc/docker/dtr/dtr.example.com.crt' is not a valid location. The certificate should be in the /etc/docker/certs.d directory, not the /etc/docker/dtr directory1.

* Passing -- insecure-registry to the Docker client is not a recommended option. This flag disables the TLS verification for the registry, which makes the communication insecure and vulnerable to attacks.

* Use self-signed certificates | Docker Docs

* Test an insecure registry | Docker Docs

* Add TLS certificates as a trusted root authority to the host OS | Docker Docs

* docker | Docker Docs

* [Deploy a registry server | Docker Docs]

Question 4

Question Type: **MultipleChoice**

Is this a way to configure the Docker engine to use a registry without a trusted TLS certificate?

Solution: Set IGNORE_TLS in the 'daemon.json' configuration file.

Options:

A- Yes

B- No

Answer:

B

Explanation:

= This is not a way to configure the Docker engine to use a registry without a trusted TLS certificate. There is no such option as IGNORE_TLS in the daemon.json configuration file.The daemon.json file is used to configure various aspects of the Docker engine, such as logging, storage, networking, and security1.To use a registry without a trusted TLS certificate, you need to either add the certificate to the trusted root certificates of the system, or configure the Docker engine to allow insecure registries2.To add the certificate to the trusted root certificates, you need to copy the certificate file to the /etc/docker/certs.d/<registry-hostname>/ directory on every Docker host2.To configure the Docker engine to allow insecure registries, you need to add the registry hostname or IP address to the "insecure-registries" array in the daemon.json file3. For example:

```
{ "insecure-registries" : ["myregistry.example.com:5000"] }
```

Note that using insecure registries is not recommended, as it exposes you to potential man-in-the-middle attacks and data corruption3.You should always use a registry with a trusted TLS certificate, or use Docker Content Trust to sign and verify your images4.Reference:

Daemon configuration file | Docker Docs

Verify repository client with certificates | Docker Docs

Test an insecure registry | Docker Docs

Content trust in Docker | Docker Docs

Question 5

Question Type: **MultipleChoice**

In Docker Trusted Registry, is this how a user can prevent an image, such as 'nginx:latest', from being overwritten by another user with push access to the repository?

Solution: Tag the image with 'nginx:immutable'.

Options:

A- Yes

B- No

Answer:

B

Explanation:

: Tagging the image with 'nginx:immutable' isnothow a user can prevent an image, such as 'nginx:latest', from being overwritten by another user with push access to the repository. Tagging the image with 'nginx:immutable' will only create a new tag for the image, but it will not prevent the original tag from being overwritten.To prevent an image tag from being overwritten in Docker Trusted Registry, the user can use the DTR web UI to make the tag immutable1.This will prevent anyone from pushing a new image with the same tag, even if they have push access to the repository1.Alternatively, the user can also use the DTR API to make the tag immutable2.Reference:Prevent tags from being overwritten), DTR API reference)

Question 6

Question Type: **MultipleChoice**

In Docker Trusted Registry, is this how a user can prevent an image, such as 'nginx:latest', from being overwritten by another user with push access to the repository?

Solution: Keep a backup copy of the image on another repository.

Options:

A- Yes

B- No

Answer:

B

Explanation:

: = Keeping a backup copy of the image on another repository is not how a user can prevent an image, such as 'nginx:latest', from being overwritten by another user with push access to the repository. This approach does not prevent the original image from being overwritten, it only provides a way to restore it from another source. However, this may not be reliable or efficient, as the backup repository may not be in sync with the original one, or may not be accessible at all times.To prevent an image from being overwritten by another user, the user can use the DTR web UI to make the tag immutable1. This feature allows the user to lock a specific tag, so that no one can push a new image with the same tag to the repository.This ensures that the image is always consistent and secure1.Reference:

Make a tag immutable | Docker Docs

Question 7

Question Type: MultipleChoice

Will this sequence of steps completely delete an image from disk in the Docker Trusted Registry?

Solution. Delete the image and remove permissions to the repository in the Docker

Trusted Registry.

Options:

A- Yes

B- No

Answer:

B

Explanation:

= The sequence of steps will not completely delete an image from disk in the Docker Trusted Registry. Deleting the image and removing permissions to the repository will only remove the image from the registry's user interface and prevent unauthorized access to it.However, the image data will still remain on the registry's storage backend until garbage collection is performed1.Garbage collection is a process that removes unused blobs (layers) from the registry's storage2.To run garbage collection, the registry must be stopped and the commandbin/registry garbage-collect /etc/docker/registry/config.ymlmust be executed3.Alternatively, the registry can be configured to run garbage collection automatically at regular intervals4.Reference:

Deleting images | Docker Documentation

Garbage collection | Docker Documentation

How to delete images from a private docker registry? | Stack Overflow

Automating garbage collection | Docker Documentation

Question 8

Question Type: **MultipleChoice**

Your organization has a centralized logging solution, such as Splunk.

Will this configure a Docker container to export container logs to the logging solution?

Solution. docker run -- log driver=splunk for every container at run time

Options:

A- Yes

B- No

Answer:

A

Explanation:

The commanddocker run --log-driver=splunkfor every container at run time will configure a Docker container to export container logs to the logging solution.The reason is that the--log-driveroption specifies the logging driver for the container, which determines how the container logs are handled1.Thesplunklogging driver is a plugin that sends container logs to HTTP Event Collector in Splunk Enterprise and Splunk Cloud2.To use thesplunklogging driver, you also need to provide some additional options with the--log-optflag, such as the Splunk token, URL, source, sourcetype, index, etc2. For example, to run a container with thesplunklogging driver and send the logs to a Splunk instance with the URLhttps://splunk.example.com:8088and the token176fabb6-7811-4b3a-8ba0-4d49302e50f2, you can use:

```
docker run --log-driver=splunk --log-opt splunk-token=176fabb6-7811-4b3a-8ba0-4d49302e50f2 --log-opt splunk-
url=https://splunk.example.com:8088 ...
```

This way, you can configure a Docker container to export container logs to Splunk, which is a centralized logging solution.Alternatively, you can also configure thesplunklogging driver as the default logging driver for the Docker daemon by setting thelog-driverandlog-optskeys in thedaemon.jsonfile and restarting Docker3. This will apply thesplunklogging driver to all containers unless overridden by the--log-driveroption.Reference:

Configure logging drivers

Splunk logging driver

Set the logging driver for the Docker daemon

Question 9

Question Type: **MultipleChoice**

Does this describe the role of Control Groups (cgroups) when used with a Docker container?

Solution: isolation between resources used by containers

Options:

A- Yes

B- No

Answer:

A

Explanation:

Control Groups (cgroups) are a Linux kernel feature that allow you to limit, modify, or allocate resources as needed1.Docker uses cgroups to isolate the resources used by containers, such as CPU, memory, disk I/O, network, etc2.This means that each container can have its own set of resource limits and constraints, and that the containers cannot interfere with each other or with the host system2.

This improves the security, performance, and reliability of the containers and the system as a whole.Reference:

Lab: Control Groups (cgroups) | dockerlabs

Docker run reference | Docker Docs

Question 10

Question Type: **MultipleChoice**

Seven managers are in a swarm cluster.

Is this how should they be distributed across three datacenters or availability zones?

Solution: 4-2-1

Options:

A- Yes

B- No

Answer:

B

Explanation:

= This is not how the seven managers should be distributed across three datacenters or availability zones.A swarm cluster is a group of Docker hosts that are running in swarm mode and act as managers or workers1.A manager node is responsible for maintaining the swarm state and orchestrating the services2.A swarm cluster needs a quorum of managers to operate, which means a majority of managers must be available and able to communicate with each other3.

The problem with distributing the seven managers as 4-2-1 is that it creates a split-brain scenario, where the cluster can lose the quorum if one datacenter or availability zone fails. For example, if the datacenter with four managers goes down, the remaining three managers will not have enough votes to form a quorum, and the cluster will stop functioning.Similarly, if the datacenter with one manager goes down, the cluster will lose the tie-breaking vote and will not be able to elect a leader4.

A better way to distribute the seven managers across three datacenters or availability zones is to use 3-2-2, which ensures that the cluster can tolerate the failure of any one datacenter or availability zone and still maintain the quorum. For example, if the datacenter with three managers goes down, the remaining four managers will have enough votes to form a quorum and elect a leader.Similarly, if the datacenter with two managers goes down, the remaining five managers will have enough votes to form a quorum and elect a leader4.Reference:

Swarm mode overview | Docker Docs

Administer and maintain a swarm of Docker Engines | Docker Docs

Raft consensus in swarm mode | Docker Docs

Docker Swarm: How to distribute managers across availability zones? - Stack Overflow

Question 11

Question Type: MultipleChoice

Is this statement correct?

Solution: A Dockerfile stores the Docker daemon's configuration options.

Options:

A- Yes

B- No

Answer:

B

Explanation:

The statement isnotcorrect. A Dockerfile does not store the Docker daemon's configuration options.A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image1. A Dockerfile is used to build images, not to configure the Docker daemon.The Docker daemon's configuration options are stored in a JSON file, which is usually located at /etc/docker/daemon.json on Linux systems, or C:\ProgramData\docker\config\daemon.json on Windows2.The JSON file allows you to customize the Docker daemon's behavior, such as enabling debug mode, setting TLS certificates, or changing the data directory2.Reference:Dockerfile reference),Docker daemon configuration overview)

Question 12

Question Type: MultipleChoice

An application image runs in multiple environments, with each environment using different certificates and ports.

Is this a way to provision configuration to containers at runtime?

Solution: Create images that contain the specific configuration for every environment.

Options:

A- Yes

B- No

Answer:

B

Explanation:

= Creating images that contain the specific configuration for every environment is not a way to provision configuration to containers at runtime.This approach violates the principle of separating application code from configuration, and makes the images less portable and reusable across different environments1.It also increases the maintenance overhead and the risk of configuration drift, as any change in the configuration would require rebuilding and redeploying the images2.To provision configuration to containers at runtime, you should use a different mechanism, such as environment variables, command-line arguments, or config maps345.Reference:

Configuration management with Containers | Kubernetes

Environment variables in Compose | Docker Docs

Override the default command | Docker Docs

Configuration management with Containers | Kubernetes