Exam Session - Preview Exam: Docker Certified Associate (DCA)

cloudacademy.com/quiz/exam/3761430/results

#1

The ______________________ allows for each namespace to have its own set of process IDs.

✓

pid namespace

✗

ipc namespace

✗

net namespace

✗

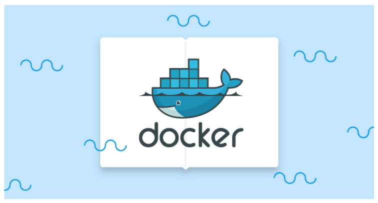
uts namespace

Explanation

Docker uses pid to handle process isolation, this means that each namespace has its own set of process IDs.

/course/introduction-to-docker-2/the-docker-architecture-1/

Covered in this lecture
The Docker Architecture
Course:Introduction to Docker



7m

#2

What is an ideal use case for Docker's bind mount storage?

✓

sharing config files from the host to the container(s)

✗

sensitive data specific to the container

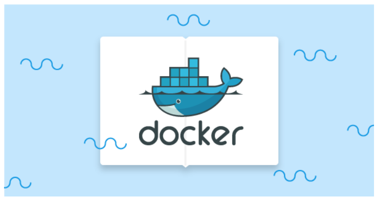
✗
sharing data among multiple running containers

✗
storing data with a remote host or cloud provider

Explanation

Volumes are recommended instead of bind mounts in most cases, but there are still use cases where bind mounts may be a superior option. The upside of bind mounts is they could work well for development, because you don't need to rebuild the image to access the new source code, so you make changes to your source and it reflects immediately. Tempfs is recommended for sensitive data related to the specific container, and volumes are recommended for sharing data among multiple running containers and storing data with a remote host or separate cloud provider, such as AWS, Azure or GCP.

/course/introduction-to-docker-2/volumes-2/
Covered in this lecture
Introduction to Persistent Storage Options in Docker
Course:Introduction to Docker



12m

#3

Which network, created upon Docker installation, does the Docker daemon automatically connect to?

✓
bridge

✗
none

✗
base

✗
host

Explanation

The `bridge` network represents the `docker0` network present in all Docker installations. Unless you specify otherwise with the `docker run --network=<NETWORK>` option, the Docker daemon connects containers to this network by default. You can see this bridge as part of a host's network stack by using the `ifconfig` command on the host.

https://docs.docker.com/engine/userguide/networking/#default-networks

#4

What is the difference between creating an image using Dockerfile and creating one using the docker commit command?

✓

Dockerfile creates an image essentially from code, while the commit command creates one based on one of your existing containers.

✕

Dockerfile creates an image based on an existing container, while the commit command creates one essentially from code.

✕

Dockerfile creates on based on an existing container, while the commit command downloads an image from Docker Hub.

✕

Using Dockerfile downloads an image from Docker Hub, while the commit command creates one based on one of your existing containers.

Explanation

When you use the Dockerfile you get to treat that image as code in a sense. You can also use the **docker commit** command to commit changes that you've made to an existing container to an image. Now, my recommendation is that you should use the Dockerfile when creating images.

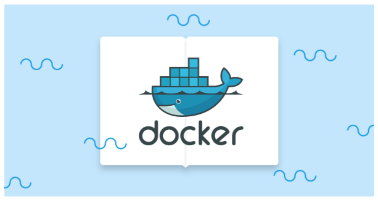
/course/introduction-to-docker-2/images-from-containers-1/

Covered in this lecture
Images From Containers
Course:Introduction to Docker

7m

#5

What of the following statements is correct regarding containers and virtual machines?



✗
It takes longer to build, test, and deploy features on containers than on virtual machines.

✓
Containers have a faster runtime than virtual machines.

✗
Containers require more troubleshooting for dependencies and version control than virtual machines

✗
Microservices must be designed using containers instead of virtual machines.

Explanation

When you are comparing to general options like containers vs. virtual machines, you can run into trouble with absolute statements, but is arguably correct that containers have a faster runtime than virtual machines because they are lighter, and have less overhead.

The other statements, whether in favor of containers or virtual machines, are not correct. You can build microservices on either one. It doesn't take longer to build, test or deploy with containers - usually the opposite is true. Finally, containers do not require more troubleshooting for dependencies or version control. Using containers is a way to mitigate that problem.

/course/introduction-to-containers/why-use-containers-1/
Covered in this lecture
Why Use Containers?
Course:Introduction to Containers

5m

#6

In regard to Docker Compose, which statement regarding volume root keys is true?

✗

You cannot use volumes in the service's configuration without a volumes key in your Compose file.

✗

External volumes for a service are only specified in Compose CLI, not in Compose file volume keys.

✗

All volume file pathways are determined by Docker, which manages volume storage.

✓

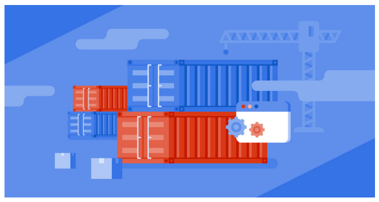
You can configure a custom volume driver using the driver and driver_opts keys.

Explanation

You can configure a custom volume driver using the driver and driver_opts keys. If you want to specify any driver-specific options, you can do so under the driver_opts key.

All the other statements are false, You can use volumes in a service configuration without a volume key. You can specify external volumes in a Compose File volume key. You can specify a volume file pathway, even though Docker manages volume storage.

/course/managing-applications-with-docker-compose/anatomy-of-a-compose-file-1/
Covered in this lecture
How to Create Docker Compose Files Using YAML
Course:Managing Applications with Docker Compose



26m

#7

When a Docker container is deleted, what happens to any data written to the container that is not stored in a data volume?

✓

it is deleted

✗

it is stored in the Docker home directory as-is

✗

it is compressed and stored in the Docker home directory

✗
it remains in the same place, waiting for the container to be re-initialized

Explanation

Data volumes and the storage driver

When a container is deleted, any data written to the container that is not stored in a *data volume* is deleted along with the container.

https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#data-volumes-and-the-storage-driver
#8

In Docker Compose, which root elements are optional? (Choose 2 answers)

✓
volumes

✓
networks

✗
services

✗
version

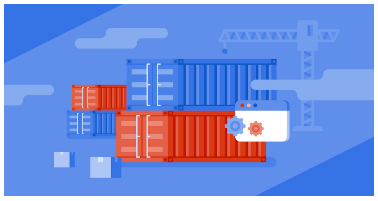
Explanation

The volumes root element is an optional key. You use the volume mapping in a way that is similar to how you use docker volume create. Services can reference volumes in each service's volumes configuration key.

Docker Compose automatically sets up a single network for your application(s) by default, adding each container for a service to the default network. You can specify your own networks but it is not required to have a networks key at the root of a Compose file.

/course/managing-applications-with-docker-compose/anatomy-of-a-compose-file-1/
Covered in this lecture
How to Create Docker Compose Files Using YAML

Course:Managing Applications with Docker Compose



26m

#9

Which of the following is NOT an advantage of using a Docker container over a virtual machine?

✗
we no longer have to worry about system dependencies

✗
resource density is increased

✗
orchestrating scaling behavior is simpler

✓
easy access to host kernel operations

Explanation

The unit of scale being an individual, portable executable has vast implications. It means that CI/CD can push updates to one part of a distributed application, system dependencies aren't a thing you worry about, resource density is increased, and orchestrating scaling behavior is a matter of spinning up new executables, not new VM hosts.

https://docs.docker.com/get-started/#conclusion

#10

Which statement regarding Docker's port mapping functionality is incorrect?

✗
You can map ports from containers to the hosts running those containers.

✗
Port mapping can be done dynamically using the publish all flag (-P).

✗
Port mapping can be done explicitly using the publish flag (-p).

✓

By default, the EXPOSE instruction in a Dockerfile performs port mapping.

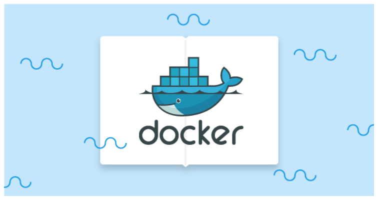
Explanation

Docker allows you to map ports from containers to the hosts running the containers. The mapping can be done dynamically with the publish all flag or explicitly with the publish flag. And by default, the expose instruction in the Docker file doesn't actually perform any port mapping.

It is up to you to determine how you want to publish the ports.

/course/introduction-to-docker-2/port-mapping-1/
Covered in this lecture
Port Mapping
Course:Introduction to Docker

8m



#11

Which statement about how Docker Compose combines multiple Compose files is correct?

✗

Specify which compose file is the base configuration within CLI to overwrite it.

✗

Identical structure are required for the base configuration and additional files to merge successfully.

✓

The -f option allows you to specify multiple non-default overrides.

✗

To check what the final overwrite is before initializing it, use the Compose config command.

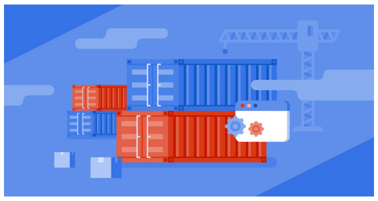
Explanation

The semantics of combining Compose files is to treat the first file as a base configuration and each additional file overrides configuration specified in the base configuration. The overrides can add configuration that isn't present in the base configuration as well, not only strictly overriding existing values in the base configuration.

The -f Compose option can be used multiple times to specify non-default override files. Each override file overriding the previous ones.

The Docker Compose config command is useful when writing and debugging multiple Compose files. It will display the effective Compose file after everything is combined.

/course/managing-applications-with-docker-compose/extending-compose-1/
Covered in this lecture
How Compose Handles and Combines Multiple Files
Course:Managing Applications with Docker Compose



9m

#12

What is the difference between the docker container prune command and the docker container rm command?

✓

The docker container prune command removes all stopped containers, while docker container rm removes only specified containers.

✕

The docker container prune command removes all containers, while docker container rm removes specified containers.

✗

The docker container prune command removes all containers, while docker container rm removes all stopped containers.

✗

The docker container prune command only removes specified containers, while docker container rm removes all stopped containers.

Explanation

If you want to remove all of the stopped containers use the Docker container prune command, but if you don't want to remove all of your stopped containers, then don't use this command. Instead, use the Docker rm command, and specify the containers that you want to remove.

/course/introduction-to-docker-2/images-from-the-dockerfile-1/
Covered in this lecture
Images From The Dockerfile
Course:Introduction to Docker

10m

#13

An individual Docker Engine instance in a Swarm is called a ________________________.

✓

node

✗

container

✗

machine

✗
an agent

Explanation

A swarm consists of one or more Docker Engines running in swarm mode. Each instance of the Docker Engine in the swarm is referred to as a node. It is possible to run multiple nodes on a single machine. For example, by using virtual machines. In production environments, you should use multiple machines to ensure availability of the swarm if a machine goes down.

https://docs.docker.com/engine/swarm/key-concepts/#nodes
Covered in this lecture
Overview
Course:Container Orchestration With Docker Swarm Mode

7m

#14

In Docker, what is defined as a runtime instance of an image?

✗
shard

✓
container

✗
cluster

✗
HVM

Explanation

A **container** is a runtime instance of an image – what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.

https://docs.docker.com/get-started/#a-brief-explanation-of-containers
Covered in this lecture

The Docker Architecture
Course:Introduction to Docker

7m

#15

Which statement is incorrect regarding the rolling update parameters you can set within Docker Swarm?

✗
The parallelism parameter sets the number of tasks updated at a time.

✗
The delay parameter sets the amount of time between updating sets of tasks.

✗
The failure action parameter determines what action to take if an update fails. It can be set to pause, continue, or automatically rollback.

✓
The failure rate parameter determines the ratio of failed updates your Swarm will tolerate. It is managed by an integrated Docker Swarm algorithm.

Explanation

With the failed update ratio parameter, you can set a ratio for the number of failed task updates to tolerate before failing a service update, and set the frequency for monitoring for a failure.

https://docs.docker.com/engine/reference/commandline/service_update/#extended-description

#16

Comparing how virtual machines and containers both run on host infrastructure, which layers does a virtual machine include that a Docker container does not? (Choose 2 answers)

✓

a hypervisor

✓

a guest operating system

✗

a host operating system

✗

binaries and libraries

Explanation

Virtual machines sharing the same hardware all share a hypervisor, and then a guest operating system within the VM is built on top of that hypervisor. Docker containers share a host operating system, and each container includes its own binaries, libraries, and application layer.

/course/introduction-to-containers/what-are-containers-1/
Covered in this lecture
What are Containers?
Course:Introduction to Containers

7m

#17

What two choices are centralized locations to download Docker Images? (Choose 2 answers)

✗

Docker Repo

✓

Docker Hub

✓

Docker Store

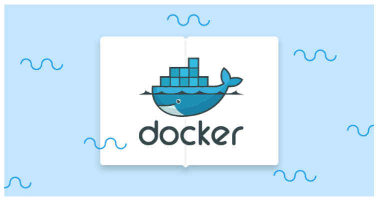
✗

Docker Cloud Registry

Explanation

Both the Docker Hub and Docker Store serve as centralized locations for Docker images to be downloaded.

/course/introduction-to-docker-2/first-container-1/
Covered in this lecture
Creating and Executing Your First Container Using Docker
Course:Introduction to Docker



5m

#18

Looking at Docker's architecture from a high level, the ____________________ represents the client.

✗

Docker daemon

✓

Docker binary

✗

application running within the Docker container

✗

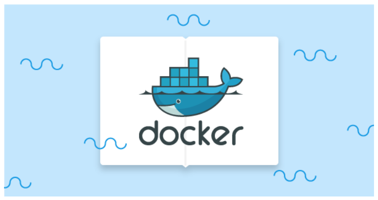
Docker image and network

Explanation

The client is the Docker binary. Whenever you use the docker command, that's the client. The diagram gives a glimpse of how the client and daemon interact together. For example, you can see that the subcommands issued by the client are sent over to the daemon, the docker pull command here instructs the daemon to get an image from the registry.

/course/introduction-to-docker-2/the-docker-architecture-1/

Covered in this lecture
The Docker Architecture
Course:Introduction to Docker



7m

#19

Which Docker image tag will give you the most up-to-date version of the image?

✗
current

✓
latest

✗
recent

✗
newest

Explanation

Specify the image version the service should use

When you create a service without specifying any details about the version of the image to use, the service uses the version tagged with the `latest` tag. You can force the service to use a specific version of the image in a few different ways, depending on your desired outcome.

https://docs.docker.com/engine/swarm/services/#specify-the-image-version-the-service-should-use
#20

What are the two types of nodes in a Docker Swarm? (Choose 2 answers)

✓
Managers

✓
Workers

✗
Agents

✗
Slaves

Explanation

Nodes can participate in a swarm by taking on specific roles: managers and workers. Every swarm requires at least one manager.

Managers have several responsibilities, but we'll start simple and consider one main responsibility. Managers accept specifications from users and drive the actual state of the swarm to the specified desired state. They do so by delegating units of work to workers in the swarm. Workers are primarily responsible for running the delegated units of work. Workers also run an agent which reports back to managers on the status of their work. A node can be either a manager or a worker.

https://docs.docker.com/engine/swarm/key-concepts/#nodes
Covered in this lecture
Overview
Course:Container Orchestration With Docker Swarm Mode

7m