Docker Containers and Concepts Quiz (100 Questions)

DevOps Learning Module

This quiz covers fundamental concepts related to Docker, containers, and orchestration. Choose the best answer for each question.

1. What is a Docker "Image"?

   A. A running instance of an application.

   B. A read-only template used to create containers.

   C. A configuration file written in YAML.

   D. A virtual machine.

   Answer: B
   Explanation: An image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and settings.

2. What is a Docker "Container"?

   A. A read-only template.

   B. A source code repository.

   C. A running, writable instance of a Docker image.

   D. The Docker daemon.

   Answer: C
   Explanation: A container is a live, running instance created *from* an image. It is the isolated environment where the application executes.

3. What command is used to build a Docker image from a Dockerfile?

   A. `docker create .`

   B. `docker build -t my-image .`

   C. `docker init .`

   D. `docker make -f Dockerfile`

   Answer: B
   Explanation: `docker build` is the command. The `-t my-image` flag "tags" the image with a name, and `.` specifies the build context (the current directory).

4. What is the name of the default file `docker build` looks for?

A. `docker.yml`

B. `build.docker`

C. `Dockerfile`

D. `Containerfile`

Answer: C
Explanation: By default, `docker build .` looks for a file named `Dockerfile` in the build context (the `.` directory).

5. Which `Dockerfile` instruction specifies the base image?

A. `BASE`

B. `FROM`

C. `RUN`

D. `IMAGE`

Answer: B
Explanation: The `FROM` instruction *must* be the first instruction in a Dockerfile. It sets the base image (e.g., `FROM ubuntu:22.04`) for subsequent instructions.

6. Which `Dockerfile` instruction executes a command *during the build process*?

A. `RUN`

B. `CMD`

C. `ENTRYPOINT`

D. `EXEC`

Answer: A
Explanation: `RUN` is used to execute commands *inside* the image at build time, creating a new layer. This is used for tasks like `RUN apt-get update` or `RUN npm install`.

7. Which `Dockerfile` instruction provides the default command to run *when the container starts*?

A. `RUN`

B. `CMD`

C. `START`

D. `EXEC`

Answer: B
Explanation: `CMD` sets the default command and/or parameters. This command is executed when the container starts, but can be easily overridden from the `docker run` command line.

8. What is the difference between `CMD` and `ENTRYPOINT`?

A. `CMD` runs at build time, `ENTRYPOINT` runs at runtime.

B. `ENTRYPOINT` configures the container to run as an executable. Arguments passed to `docker run` are appended to the `ENTRYPOINT`.

C. `CMD` is the only one that can be overridden.

D. There is no difference.

Answer: B
Explanation: `ENTRYPOINT` sets the main command for the container. `CMD` is then used to specify the *default arguments* for that `ENTRYPOINT`.

9. Which command is used to run a container from an image?

A. `docker create ubuntu`

B. `docker start ubuntu`

C. `docker run ubuntu`

D. `docker exec ubuntu`

Answer: C
Explanation: `docker run` is the command to create and start a container from an image in one step (e.g., `docker run -it ubuntu bash`).

10. What does the `-d` flag do in `docker run`?

A. Deletes the container after it exits.

B. Runs the container in "detached" (background) mode.

C. Runs the container in "development" mode.

D. Disables networking.

Answer: B
Explanation: `-d` (detached) runs the container in the background and prints the new container ID.

11. What do the `-it` flags do in `docker run`?

A. `-i` (interactive) keeps STDIN open, and `-t` allocates a pseudo-TTY (a terminal).

B. `-i` (image) and `-t` (tag).

C. `-i` (install) and `-t` (test).

D. `-i` (internal) and `-t` (temporary).

Answer: A
Explanation: You use `-it` (e.g., `docker run -it ubuntu bash`) to get an interactive shell inside the container.

12. What command lists all *running* containers?

A. `docker ls`

B. `docker images`

C. `docker ps`

D. `docker list -containers`

Answer: C
Explanation: `docker ps` (process status) lists all currently running containers.

13. What command lists *all* containers, including stopped ones?

    A. `docker ps -a`

    B. `docker ps -all`

    C. `docker list all`

    D. Both A and B.

    Answer: D
    Explanation: The `-a` or `-all` flag shows all containers, including those that have exited.

14. What command lists all Docker images on your local machine?

    A. `docker ps -i`

    B. `docker images`

    C. `docker list images`

    D. `docker img ls`

    Answer: B
    Explanation: `docker images` (or `docker image ls`) lists all images stored locally.

15. What command is used to delete a stopped container?

    A. `docker del <container_id>`

    B. `docker rmi <container_id>`

    C. `docker rm <container_id>`

    D. `docker container prune`

    Answer: C
    Explanation: `docker rm` (remove) is used to delete one or more containers.

16. What command is used to delete a Docker image?

    A. `docker rm <image_id>`

    B. `docker rmi <image_id>`

    C. `docker del -image <image_id>`

    D. `docker image prune`

    Answer: B
    Explanation: `docker rmi` (remove image) is used to delete images. You cannot delete an image if it is being used by a container.

17. What is the "Docker Hub"?

    A. The Docker daemon.

    B. The default, public "Registry" (repository) for Docker images.

    C. A networking driver.

    D. A Docker command.

    Answer: B
    Explanation: Docker Hub is the default, cloud-hosted registry where you can find and share public images (like `ubuntu`, `nginx`, etc.)

18. What command downloads an image from a registry (like Docker Hub)?

    A. `docker fetch <image>`

    B. `docker pull <image>`

    C. `docker download <image>`

    D. `docker run <image>`

    Answer: B
    Explanation: `docker pull ubuntu` will download the latest `ubuntu` image. (`docker run` will also do this implicitly if the image is not found locally).

19. What command uploads an image to a registry?

    A. `docker push <image>`

    B. `docker send <image>`

    C. `docker upload <image>`

    D. `docker hub upload <image>`

    Answer: A
    Explanation: Before you can push, you must first "tag" your image with the registry's name (e.g., `docker tag my-image user/my-image`), and then `docker push user/my-image`.

20. What is a "Docker Volume"?

    A. A Docker image.

    B. A way to persist data, managed by Docker, that exists outside the container's writable layer.

    C. The size of a container.

    D. A network mount.

    Answer: B
    Explanation: Volumes are the preferred way to persist data (like a database). They are managed by Docker and stored on the host, but are decoupled from the container's lifecycle.

21. What is a "Bind Mount"?

    A. The same as a volume.

    B. A way to mount a specific file or directory from the host machine into the container.

    C. A way to mount a network share.

    D. A read-only volume.

    Answer: B
    Explanation: Bind mounts (e.g., `docker run -v /path/on/host:/path/in/container`) link a host path directly into the container. This is common for development (mounting source code).

22. What is the `Dockerfile` instruction `COPY` used for?

    A. To copy a file from a URL.

    B. To copy a file from a `.zip` archive.

    C. To copy files or directories from the build context (your local machine) into the image.

    D. To copy files between build stages.

    Answer: C
    Explanation: `COPY . /app` is a common instruction to copy the application code from the host's build directory into the `/app` directory in the image.

23. What is the difference between `COPY` and `ADD` in a Dockerfile?

    A. There is no difference.

    B. `ADD` can copy files from a URL and automatically extract `.tar.gz` archives.

    C. `COPY` is deprecated.

    D. `ADD` is faster.

    Answer: B
    Explanation: `ADD` has "magic" features (like URL download and tar extraction) that `COPY` does not. Best practice is to *prefer `COPY`* unless you specifically need those magic features.

24. Which `Dockerfile` instruction sets the default working directory for subsequent `RUN`, `CMD`, and `ENTRYPOINT` instructions?

    A. `PATH`

    B. `DIR`

    C. `WORKDIR`

    D. `CWD`

    Answer: C
    Explanation: `WORKDIR /app` is like running `cd /app`. All following instructions will be executed from within that directory.

25. What is the `EXPOSE` instruction used for?

    A. To publish a port to the host.

    B. To open a port in the container's firewall.

    C. To document which port the application inside the container is intended to listen on.

    D. To connect to another container.

    Answer: C
    Explanation: `EXPOSE 8080` is *only* documentation. It does *not* actually open or publish the port. You still must use `docker run -p 80:8080` to publish it.

26. How do you publish a container's port (e.g., 80) to a port on the host (e.g., 8080)?

    A. `docker run -p 80:8080`

    B. `docker run -p 8080:80`

    C. `docker run -expose 8080:80`

    D. `docker run -port 80:8080`

    Answer: B
    Explanation: The format is `-p <host_port>:<container_port>`. This command maps port 8080 on the host to port 80 inside the container.

27. What is a "Docker layer"?

    A. A Docker network.

    B. Each instruction in a Dockerfile creates a read-only "layer" in the image.

    C. A Docker volume.

    D. A running container.

    Answer: B
    Explanation: Docker images are made of a stack of read-only layers. This is what makes builds efficient. If a layer hasn't changed, Docker reuses it from the cache.

28. How does Docker's layer caching work?

    A. It doesn't cache layers.

    B. If an instruction (and its files) in a Dockerfile hasn't changed, Docker reuses that layer and all subsequent layers.

    C. If an instruction (and its files) in a Dockerfile hasn't changed, Docker reuses that layer from the cache.

    D. If a `RUN` command changes, Docker re-runs it, but reuses the layers *after* it.

    Answer: C
    Explanation: Docker reuses layers as long as the instruction *and* the files it depends on (e.g., in a `COPY`) are unchanged. Once one layer is rebuilt, all subsequent layers are also rebuilt.

29. What is "Docker Compose"?

    A. A tool for building images.

    B. A tool for defining and running multi-container Docker applications.

    C. A tool for managing a single container.

    D. A tool for managing Docker Swarm.

    Answer: B
    Explanation: Docker Compose is used for development and testing. It allows you to define a multi-service application (e.g., a web app, a database, a cache) in a single YAML file.

30. What is the default filename for Docker Compose?

    A. `docker.yml`

    B. `compose.yml`

    C. `docker-compose.yml`

    D. `Dockerfile`

    Answer: C
    Explanation: By default, `docker-compose` commands (like `docker-compose up`) look for a file named `docker-compose.yml`.

31. What command is used to start all services defined in a `docker-compose.yml` file?

    A. `docker-compose run`

    B. `docker-compose start`

    C. `docker-compose up`

    D. `docker-compose exec`

    Answer: C
    Explanation: `docker-compose up` builds, (re)creates, starts, and attaches to containers for a service. Add `-d` to run in detached mode.

32. What command is used to stop and remove all services defined in a `docker-compose.yml` file?

    A. `docker-compose rm`

    B. `docker-compose stop`

    C. `docker-compose down`

    D. `docker-compose kill`

    Answer: C
    Explanation: `docker-compose down` stops and removes the containers. It can also remove networks and volumes if specified (`-v`).

33. In `docker-compose.yml`, what is a "service"?

A. A network.

B. A volume.

C. A definition of a single container in the application (e.g., `web`, `db`).

D. The host machine.

Answer: C
Explanation: A "service" in Compose defines one container, including its image, ports, volumes, and network connections.

34. What is the `ENV` instruction in a Dockerfile used for?

A. To set environment variables *during the build*.

B. To set environment variables that will exist *inside the running container*.

C. To set environment variables for the Docker daemon.

D. Both A and B.

Answer: D
Explanation: `ENV MY_VAR=foo` sets an environment variable that is available for subsequent `RUN` instructions *and* for the final running container.

35. What command allows you to run a command *inside* an already running container?

A. `docker run <container_id> <command>`

B. `docker attach <container_id>`

C. `docker exec -it <container_id> <command>`

D. `docker start <container_id> <command>`

Answer: C
Explanation: `docker exec` is used to "execute" a new command inside a running container. For example, `docker exec -it my-app bash` gives you a new shell inside the `my-app` container.

36. What command allows you to view the logs of a running container?

A. `docker exec <container_id> cat /var/log/app.log`

B. `docker logs <container_id>`

C. `docker show-logs <container_id>`

D. `docker tail <container_id>`

Answer: B
Explanation: `docker logs` fetches the `stdout` and `stderr` streams from a container. Use `-f` to "follow" the logs in real-time.

37. What is the `-rm` flag used for in `docker run`?

A. To remove the image after running.

B. To automatically remove the container (delete it) when it exits.

C. To run the container as the `rm` user.

D. To mount a read-only volume.

Answer: B
Explanation: This is a very useful flag for temporary or test containers. It prevents you from having to `docker rm` the container manually after it stops.

38. What is the "bridge" network in Docker?

A. The default, private network that containers are connected to.

B. A network that connects directly to the host's network.

C. A network for swarm mode.

D. A tool for connecting to remote containers.

Answer: A
Explanation: The "bridge" network is a private, internal network on the host. Containers on the same bridge network can communicate with each other, and Docker manages port forwarding.

39. What is the "host" network in Docker?

A. The default bridge network.

B. A network that gives the container its own IP on the LAN.

C. It removes network isolation; the container shares the host's network stack.

D. A network for connecting to the Docker Hub.

Answer: C
Explanation: `docker run -net=host` means the container does *not* get its own IP. It binds ports directly to the host machine's network interface.

40. What is a "multi-stage build"?

A. A Dockerfile that uses multiple `FROM` instructions to separate the build-time environment from the final runtime image.

B. A Docker Compose file with multiple services.

C. A build that runs in multiple stages (build, test, deploy).

D. A build that creates multiple images.

Answer: A
Explanation: This is a key optimization. You use a large "builder" image (e.g., `FROM golang AS builder`) to compile, then a small "runtime" image (e.g., `FROM alpine`) and `COPY -from=builder ...` just the compiled artifact.

41. What is the `docker inspect` command used for?

A. To run security scans on an image.

B. To get detailed, low-level information (in JSON) about a container or image.

C. To view the logs.

D. To get a shell inside the container.

Answer: B
Explanation: `docker inspect` provides a wealth of information, such as the container's IP address, volume mounts, and port mappings.

42. What is the `docker tag` command used for?

A. To add metadata to a container.

B. To create an alias (a new name, or "tag") for an existing image.

C. To add a tag to a Dockerfile.

D. To delete a tag.

Answer: B
Explanation: `docker tag my-image:latest my-registry.com/user/my-image:1.0` creates a new tag `1.0` pointing to the *same* image. This is necessary before pushing to a registry.

43. What is the `docker save` command used for?

A. To save a container's state (like a snapshot).

B. To save an image (or images) to a `.tar` archive.

C. To save the logs of a container.

D. To save Docker's configuration.

Answer: B
Explanation: `docker save -o my-image.tar my-image` is used to export an image to a file, which can then be transferred to another machine.

44. What is the `docker load` command used for?

A. To load a new configuration.

B. To load a `.tar` archive (created by `docker save`) into the local image cache.

C. To load a container from disk.

D. To load a volume.

Answer: B
Explanation: `docker load -i my-image.tar` is the companion to `docker save`, used for "sneakernet" (offline) transfer of images.

45. What is the "Docker Engine"?

A. A Dockerfile linter.

B. The `docker-compose` tool.

C. The background service (daemon, `dockerd`) that manages and runs containers.

D. The `docker` command-line client.

Answer: C
Explanation: The Docker Engine is the core, background service (a daemon) that listens for API requests (from the `docker` client) and manages containers, images, volumes, and networks.

46. What command is used to pause all processes in a container?

    A. `docker stop`

    B. `docker pause`

    C. `docker freeze`

    D. `docker exec -it <id> sleep`

    Answer: B
    Explanation: `docker pause` suspends all processes in the container. `docker unpause` resumes them.

47. What is the difference between `docker stop` and `docker kill`?

    A. `stop` is graceful; `kill` is immediate.

    B. `kill` is graceful; `stop` is immediate.

    C. `stop` deletes the container; `kill` only stops it.

    D. There is no difference.

    Answer: A
    Explanation: `docker stop` sends a SIGTERM signal, giving the container time to shut down gracefully. `docker kill` sends a SIGKILL signal, which terminates it immediately.

48. What is "Docker Swarm"?

    A. A container registry.

    B. A tool for building images.

    C. Docker's native container orchestration tool for managing a cluster of Docker nodes.

    D. A Docker networking driver.

    Answer: C
    Explanation: Docker Swarm is a simpler alternative to Kubernetes. It allows you to join multiple Docker hosts (nodes) into a "swarm" and deploy "services" across them.

49. In Docker Swarm, what is a "service"?

    A. A single container.

    B. A definition of a task, which can include multiple "replicas" (containers) running across the cluster.

    C. A "manager" node.

    D. A network.

Answer: B
Explanation: A Swarm "service" (e.g., `docker service create -replicas 3 nginx`) defines the desired state, and Swarm ensures that 3 `nginx` containers are running somewhere on the cluster.

50. In Docker Swarm, what are "manager" nodes?

    A. Nodes that only run application containers.

    B. Nodes that handle the cluster management and orchestration tasks.

    C. Nodes that store volumes.

    D. The Docker Hub.

    Answer: B
    Explanation: Manager nodes maintain the cluster state. "Worker" nodes are the nodes that simply run the containers (tasks) assigned to them by the managers.

51. What technology does Docker use for process isolation?

    A. Virtual Machines

    B. SELinux

    C. Linux Namespaces

    D. Hyper-V

    Answer: C
    Explanation: Linux Namespaces are a kernel feature that isolates a container's view of the system (e.g., PID, network, mount points).

52. What technology does Docker use for resource limiting (CPU, Memory)?

    A. AppArmor

    B. Linux Namespaces

    C. Control Groups (cgroups)

    D. Systemd

    Answer: C
    Explanation: cgroups are a Linux kernel feature that allows you to limit and account for the resources (CPU, memory, I/O) used by a group of processes.

53. What is the `ARG` instruction in a Dockerfile?

    A. It's the same as `ENV`.

    B. It defines a variable that is *only* available during the build process.

    C. It defines the `CMD` arguments.

    D. It defines the `ENTRYPOINT` arguments.

Answer: B
Explanation: `ARG` variables are build-time variables (e.g., `ARG VERSION=latest`). They are *not* available in the final running container, unlike `ENV`.

54. How do you pass a build-time argument to `docker build`?

    A. `docker build -build-arg VERSION=1.2.3 .`

    B. `docker build -e VERSION=1.2.3 .`

    C. `docker build -a VERSION=1.2.3 .`

    D. `docker build -env VERSION=1.2.3 .`

    Answer: A
    Explanation: The `-build-arg` flag is used to pass values for `ARG` instructions defined in the Dockerfile.

55. What is the `HEALTHCHECK` instruction in a Dockerfile?

    A. A command that runs `docker ps`.

    B. A command that tells Docker how to test a container to check if it is still working.

    C. A command that runs at build time to test the image.

    D. A command that runs unit tests.

    Answer: B
    Explanation: `HEALTHCHECK` defines a command (e.g., `curl -f http://localhost/ || exit 1`) that Docker will run periodically *inside* the container to check its health.

56. What is the `.dockerignore` file?

    A. The same as `.gitignore`.

    B. A file that tells the Docker daemon which files to *exclude* from the build context.

    C. A file that lists containers to ignore.

    D. A file that lists images to ignore.

    Answer: B
    Explanation: This is a critical optimization. You use `.dockerignore` (with syntax like `.gitignore`) to prevent large, unnecessary files (like `.git`, `node_modules`) from being sent to the Docker daemon.

57. What is the "build context" in Docker?

    A. The Dockerfile itself.

    B. The set of files (usually the directory `.` or a `.tar` file) sent to the Docker daemon to be used for the build.

    C. The base image.

    D. The Docker host.

Answer: B
Explanation: When you run `docker build .`, the `.` (the current directory) is the build context. The Docker client archives this directory and sends it to the daemon.

58. What is the `USER` instruction in a Dockerfile?

    A. To specify the user for the `docker build`.

    B. To set the username that subsequent `RUN`, `CMD`, and `ENTRYPOINT` instructions will run as.

    C. To add a user to the container.

    D. To specify the `docker login` user.

    Answer: B
    Explanation: This is a key security best practice. You should `RUN useradd ...` and then `USER myuser` to avoid running your application as `root`.

59. What is a "dangling image"?

    A. An image that failed to build.

    B. An image that has been pushed to Docker Hub.

    C. An image that is not tagged and is not used by any container.

    D. An image that is running.

    Answer: C
    Explanation: Dangling images often occur when you rebuild an image with the same tag. The old image loses its tag and becomes a dangling (unnamed) image, taking up disk space.

60. What command is used to remove all dangling images?

    A. `docker rmi -dangling`

    B. `docker image prune`

    C. `docker system prune`

    D. `docker rmi $(docker images -f "dangling=true" -q)`

    Answer: B
    Explanation: `docker image prune` is the modern, easy way to remove all dangling images. (D is the older, manual way).

61. What command is used to remove all stopped containers, all dangling images, and all unused networks?

    A. `docker system prune`

    B. `docker clean all`

    C. `docker reset`

    D. `docker rm -a`

Answer: A
Explanation: `docker system prune` is the main "cleanup" command. Add `-a` to also remove *all* unused images, not just dangling ones.

62. What is the `docker commit` command?

    A. The same as `git commit`.

    B. A command to create a new *image* from a *container's* current state.

    C. A command to save a Dockerfile.

    D. A command to commit changes to Docker Hub.

    Answer: B
    Explanation: This is generally discouraged. It's the "manual" way to create an image, but it's not reproducible. The best practice is to *always* use a Dockerfile.

63. What is the `ONBUILD` instruction in a Dockerfile?

    A. A command that runs when the build starts.

    B. A command that registers a "trigger" to be executed when another image uses this one as a `FROM` base.

    C. A command that runs only on `docker build`.

    D. A command that runs when the container starts.

    Answer: B
    Explanation: `ONBUILD` is a trigger. If you `FROM my-image`, and `my-image` had an `ONBUILD COPY . /app` instruction, the `COPY` command will run as part of *your* build.

64. How do you run a container with a specific, static IP address on a user-defined bridge network?

    A. You cannot set a static IP.

    B. `docker run -ip 172.18.0.22 ...`

    C. `docker run -network my-net -ip 172.18.0.22 ...`

    D. `docker run -static-ip 172.18.0.22 ...`

    Answer: C
    Explanation: You can only assign a static IP on a *user-defined* bridge network (which you create with `docker network create`), not the default `bridge` network.

65. What is the "overlay" network driver used for?

    A. For single-host bridge networking.

    B. For Docker Swarm, to create a network that spans multiple hosts.

    C. For connecting directly to the host's network.

    D. For development use.

Answer: B
Explanation: An overlay network allows containers running on different hosts (but in the same Swarm) to communicate with each other as if they were on the same private network.

66. What is the `docker top` command?

    A. It shows the top-level images.

    B. It shows the top-level containers.

    C. It's an alias for `docker ps`.

    D. It lists the running processes *inside* a container.

    Answer: D
    Explanation: `docker top <container_id>` is like running `ps` or `top` *inside* the container, but from the host.

67. What is the `docker system df` command?

    A. It's the same as the Linux `df` command.

    B. It shows a summary of Docker disk usage (images, containers, volumes).

    C. It clears the Docker cache.

    D. It shows the Dockerfile for an image.

    Answer: B
    Explanation: This command is very useful for seeing how much disk space is being taken up by images, containers, and (especially) build cache.

68. What is a "tmpfs" mount in Docker?

    A. A regular volume.

    B. A bind mount.

    C. A temporary filesystem that is mounted in the container's RAM.

    D. A read-only filesystem.

    Answer: C
    Explanation: `docker run -tmpfs /app` mounts a temporary filesystem in `/app`. It's very fast (in-memory) and is automatically destroyed when the container stops.

69. What is the `-read-only` flag for `docker run`?

    A. It makes all volumes read-only.

    B. It makes the container's entire filesystem read-only.

    C. It prevents the container from being deleted.

    D. It does not do anything.

Answer: B
Explanation: This is a great security practice. It makes the container's root filesystem read-only. You must explicitly mount volumes for any paths that *need* to be writable.

70. How do you add metadata (like an author) to a Docker image?

    A. `docker commit -author="Me"`

    B. Use the `LABEL` instruction in the Dockerfile.

    C. `docker tag -label ...`

    D. `ENV AUTHOR="Me"`

    Answer: B
    Explanation: The `LABEL` instruction is the standard way to add key-value metadata to an image (e.g., `LABEL maintainer="me@example.com"`).

71. What is `docker context`?

    A. The build context.

    B. A tool to switch the Docker client between different Docker daemons (e.g., local, remote, Docker Desktop).

    C. A security feature.

    D. A networking tool.

    Answer: B
    Explanation: `docker context` allows your single `docker` client to control multiple, different Docker environments.

72. What is the `docker attach` command?

    A. It's the same as `docker exec`.

    B. It attaches your terminal's STDIN, STDOUT, and STDERR to the *main running process* (PID 1) in the container.

    C. It attaches a volume to a container.

    D. It attaches a network to a container.

    Answer: B
    Explanation: This is different from `exec`. `attach` "hijacks" the main process. If you `attach` to a web server, you'll see its log output.

73. What is "BuildKit"?

    A. An older, deprecated Docker builder.

    B. A next-generation, high-performance builder backend for Docker.

    C. A tool for building `docker-compose.yml` files.

    D. A CI/CD tool for Docker.

Answer: B
Explanation: BuildKit is the modern, default builder. It enables features like parallel builds, better caching, and improved performance.

74. How do you enable BuildKit?

    A. `docker build -buildkit=true`

    B. It is enabled by default in modern Docker versions.

    C. Set the environment variable `DOCKER_BUILDKIT=1`.

    D. Both B and C.

Answer: D
Explanation: In new versions of Docker Desktop and Docker Engine, BuildKit is the default. For older versions, setting `DOCKER_BUILDKIT=1` enables it.

75. What is the `-squash` flag in `docker build`?

    A. It deletes the build cache.

    B. It squashes all filesystem layers from the build into a *single* new layer.

    C. It fails the build if the image is too large.

    D. It runs a `git squash` first.

Answer: B
Explanation: This is used to create a "smaller" final image by collapsing all the `RUN`, `COPY`, etc. layers into one layer, at the cost of losing cache for those layers.

76. What is `docker system prune -a -volumes`?

    A. A command to delete all images.

    B. A command to delete all volumes.

    C. A very destructive command that deletes all stopped containers, all unused networks, all dangling images, *all* unused images, and *all* unused volumes.

    D. A command to prune all containers.

Answer: C
Explanation: This is the "nuke from orbit" command. `-a` removes *all* unused images (not just dangling), and `-volumes` removes *all* local volumes not used by a container.

77. In Docker Compose, what does `depends_on` do?

    A. It downloads dependencies.

    B. It controls the startup *order* of services.

    C. It links two services' networks.

    D. It passes environment variables.

Answer: B
Explanation: If your `web` service has `depends_on: [db]`, Docker Compose will start the `db` service *before* it starts the `web` service.

78. What is the `docker stats` command?

    A. It shows disk usage statistics.

    B. It shows network statistics.

    C. It provides a live stream of resource usage (CPU, Memory) for all running containers.

    D. It shows build statistics.

    Answer: C
    Explanation: `docker stats` is like running `top`, but for containers. It's a great way to see which container is consuming all your CPU or RAM.

79. What is "rootless" Docker?

    A. A Docker daemon that cannot build images.

    B. A mode where the Docker daemon and containers run as a regular user, without `root` privileges.

    C. A Docker container that runs a non-root user.

    D. A Docker container with no filesystem.

    Answer: B
    Explanation: This is a modern security feature. It allows a user to run the `dockerd` daemon in their own user namespace, which is much more secure as it avoids giving root access to the host.

80. What is the `docker cp` command?

    A. To copy a container.

    B. To copy files or directories *between* the host and a container.

    C. To copy an image.

    D. To copy a volume.

    Answer: B
    Explanation: `docker cp <host_path> <container>:<container_path>` (and vice-versa) is used to get files in and out of a container.

81. What is the "cgroups" (Control Groups) feature used for by Docker?

    A. For process isolation (PID).

    B. For network isolation.

    C. For limiting and monitoring resource usage (CPU, RAM, I/O).

    D. For container storage.

Answer: C
Explanation: Cgroups are the Linux kernel feature that enforces resource limits. This is how `docker run -memory 512m` works.

82. What is the "namespaces" feature used for by Docker?

    A. For limiting resource usage.

    B. For isolating a container's view of the system (PID, network, mounts, users).

    C. For storing image layers.

    D. For managing the build cache.

Answer: B
Explanation: Namespaces are the isolation feature. They make a container *think* it's the only process (PID 1) on its own machine.

83. How do you pass an environment variable to a container at runtime?

    A. `docker run -e "MY_VAR=foo"`

    B. `docker run -env "MY_VAR=foo"`

    C. `docker run -v "MY_VAR=foo"`

    D. Both A and B.

Answer: D
Explanation: The `-e` or `-env` flag is used to set environment variables inside the container. This is the standard way to pass configuration (like database passwords) to an image.

84. What is the `-link` flag in `docker run`?

    A. A modern, recommended way to connect containers.

    B. A legacy, deprecated way to connect containers by editing `/etc/hosts`.

    C. A way to link a volume.

    D. A way to link an image.

Answer: B
Explanation: `-link` is a deprecated feature. The modern, recommended way to connect containers is by creating a user-defined `docker network`.

85. How do you create a user-defined bridge network?

    A. `docker network create my-net`

    B. `docker network new my-net`

    C. `docker bridge create my-net`

    D. `docker network attach my-net`

Answer: A
Explanation: `docker network create <name>` (using the default `bridge` driver) creates a new private network. Containers on this network can reach each other by name.

86. If you have a container named `db` on a network `my-net`, how can another container on the same network reach it?

    A. By finding its IP address from `docker inspect`.

    B. By using the hostname `db`.

    C. It cannot.

    D. By using `-link`.

    Answer: B
    Explanation: Docker's built-in DNS service on user-defined networks allows containers to resolve each other by their *service name* or *container name*.

87. What is an "overlay" network used for?

    A. To connect containers on *different* Docker hosts.

    B. To connect containers on the *same* host.

    C. To connect a container to the host's network.

    D. To isolate a container.

    Answer: A
    Explanation: Overlay networks are the networking solution for multi-host clustering (Docker Swarm). They create a virtual network *over* the physical host networks.

88. What is the "macvlan" network driver?

    A. A network driver that assigns a MAC address (and thus an IP) from the host's physical network to a container.

    B. A driver for Mac only.

    C. A driver for virtualizing the MAC address.

    D. The default bridge network.

    Answer: A
    Explanation: This is an advanced networking mode used when you want your container to appear as a "real" device on your physical network, with its own unique IP.

89. What is the `docker commit` command (again)?

    A. It's the recommended way to build images.

    B. It creates a new image from the *current filesystem state* of a running (or stopped) container.

    C. It commits a Dockerfile to Git.

    D. It saves a container's state to disk.

Answer: B
Explanation: This is generally bad practice ("black box" images), but can be useful for debugging. It creates an image from the container's diff layer.

90. What is the "writable container layer"?

    A. The base image.

    B. A thin, writable layer on top of the read-only image layers, where all changes made by the container are stored.

    C. A Docker volume.

    D. The build cache.

    Answer: B
    Explanation: When a container starts, Docker adds a thin, writable layer. When you "change" a file, Docker copies it from a read-only layer (copy-on-write) into this writable layer.

91. What happens to the "writable container layer" when the container is deleted?

    A. It is merged into the base image.

    B. It is archived.

    C. It is permanently deleted (unless you ran `docker commit`).

    D. It is converted to a volume.

    Answer: C
    Explanation: This is why containers are "ephemeral." All data stored in the container (not in a volume) is lost when the container is deleted.

92. What is a "named volume" (e.g., `-v my-volume:/data`)?

    A. A bind mount.

    B. A volume created and managed by Docker, referenced by a name (`my-volume`).

    C. A temporary volume.

    D. An old, deprecated feature.

    Answer: B
    Explanation: This is the modern, recommended way to persist data. Docker manages the storage location on the host, and you just reference it by name.

93. What is an "anonymous volume" (e.g., `-v /data`)?

    A. A volume that is not secure.

    B. A volume that Docker creates with a random hash name.

    C. A bind mount.

    D. A tmpfs mount.

Answer: B
Explanation: If you only specify the container path, Docker still creates a volume (to persist the data) but gives it a long, random ID. This is harder to manage.

94. What is the `docker volume ls` command?

    A. It lists all containers.

    B. It lists all volumes (both named and anonymous) managed by Docker.

    C. It lists all bind mounts.

    D. It lists all Dockerfiles.

    Answer: B
    Explanation: This command lists all the volumes Docker is aware of, allowing you to manage them (e.g., `docker volume prune`).

95. In Docker Compose, what does `build: .` mean?

    A. It specifies an image named `build`.

    B. It tells Compose to build an image from the Dockerfile in the current directory (`.`) instead of pulling one.

    C. It specifies a bind mount to the current directory.

    D. It is not valid syntax.

    Answer: B
    Explanation: Instead of `image: my-image`, you can use `build: .` to tell Compose to run `docker build .` and use the resulting image for the service.

96. What is the `docker-compose.yml` `restart: always` policy?

    A. It always restarts the service when you run `docker-compose up`.

    B. It tells the Docker daemon to always restart the container if it stops (e.g., on failure or host reboot).

    C. It forces a rebuild of the image.

    D. It always restarts the host machine.

    Answer: B
    Explanation: This is a restart policy. `restart: always` ensures the container will always be restarted by the daemon unless it is explicitly stopped.

97. What is the `docker-compose.yml` `environment` key?

    A. To set the `NODE_ENV` variable.

    B. To set environment variables inside the container for that service.

    C. To specify a `.env` file.

    D. To specify the host environment.

Answer: B
Explanation: This is the Compose equivalent of the `docker run -e` flag. You can provide a list of `KEY=VALUE` pairs.

98. What is the `.env` file used by Docker Compose?

    A. It's a file of environment variables to be passed *into* the container.

    B. It's a file of variables that are substituted *inside* the `docker-compose.yml` file.

    C. It's the Dockerfile.

    D. It's a file of build arguments.

    Answer: B
    Explanation: If you have `image: my-image:${TAG}` in your Compose file, Compose will look for a `.env` file in the same directory to find the value of `$TAG`.

99. What does "container orchestration" mean?

    A. The process of building a container.

    B. The process of automating the deployment, scaling, management, and networking of containers.

    C. The process of writing a Dockerfile.

    D. The Docker daemon.

    Answer: B
    Explanation: "Orchestration" is managing the *lifecycle* of containers at scale. Kubernetes and Docker Swarm are the two main orchestrators.

100. What is Kubernetes (K8s)?

    A. A simpler alternative to Docker Swarm.

    B. A container runtime.

    C. A powerful, open-source container orchestration platform.

    D. A CI/CD tool.

    Answer: C
    Explanation: Kubernetes is the industry-standard tool for deploying, scaling, and managing complex, containerized applications in production.