No, the pwd command **does not** work inside a **Dockerfile** because a **Dockerfile** is not an interactive shell environment. Instead, it is a set of instructions to build a Docker image.

**How to Get the Current Working Directory in a Dockerfile**

If you need to set or reference the working directory inside a **Dockerfile**, you should use the WORKDIR instruction.

**Example:**

dockerfile

FROM ubuntu:latest

WORKDIR /app

RUN pwd # This will print "/app" during the build

* The WORKDIR /app sets /app as the working directory.
* RUN pwd will execute during **image build time**, not runtime.

**Why pwd Doesn’t Work Like in a Shell**

* Inside a Dockerfile, commands are executed in temporary **layers**.
* A RUN pwd command does not persist beyond that specific RUN step.

**Example:**

dockerfile

FROM ubuntu:latest

RUN mkdir /data

RUN cd /data

RUN pwd # This still prints "/" because each RUN command runs in a new shell

**Solution:** Use WORKDIR to persist the directory change.

dockerfile

FROM ubuntu:latest

WORKDIR /data

RUN pwd # Now prints "/data"

**Alternative: Using pwd in a Running Container**

If you want to check the working directory inside a **running container**, you can use:

sh

docker run --rm ubuntu pwd

Or, if you are inside a running container:

sh

Pwd

Docker **restart policies** define the conditions under which Docker should automatically restart a container when it exits. This is useful for ensuring that containers are resilient and can recover from crashes or failures without manual intervention.

**Types of Docker Restart Policies:**

1. **no** (default)
   * **Behavior**: The container will **not be restarted** when it stops.
   * **Use Case**: When you don't want the container to restart automatically, e.g., for short-lived jobs.
2. **always**
   * **Behavior**: The container will **always be restarted** if it stops or Docker daemon restarts.
   * **Use Case**: For long-running services that should always be running, like web servers or databases.
3. **unless-stopped**
   * **Behavior**: The container will be restarted unless it is **manually stopped** by the user (via docker stop).
   * **Use Case**: Similar to always, but it allows manual control to stop the container without it being automatically restarted.
4. **on-failure**
   * **Behavior**: The container will be restarted only if it **exits with a non-zero exit code** (indicating a failure).
   * You can optionally specify a **maximum restart limit**:

sh

--restart on-failure:<max-retries>

Where <max-retries> specifies the number of times Docker will attempt to restart the container before giving up.

* + **Use Case**: Ideal for containers that can fail temporarily (e.g., an API that might need to be retried).

**Usage Examples:**

1. **Always Restart the Container:**

sh

docker run --restart always my-container

1. **Restart Only on Failure:**

sh

docker run --restart on-failure my-container

1. **Restart on Failure with Max Retries (e.g., 3 retries):**

sh

docker run --restart on-failure:3 my-container

1. **Restart Unless Manually Stopped:**

sh

docker run --restart unless-stopped my-container

**How Restart Policies Work:**

* Docker checks the container's exit status:
  + If it exits with **0** (success), no restart occurs.
  + If it exits with **non-zero** (failure), the restart policy decides whether it will restart.
* The restart policy applies only after the container has **stopped** (either due to a failure or a normal exit).

**Changing Restart Policy of an Existing Container**

You can update the restart policy of an existing container using the docker update command:

sh

docker update --restart always <container\_name\_or\_id>

**When to Use Each Policy:**

* **always**: For critical, long-running services like databases and APIs that should never stop unless explicitly stopped.
* **unless-stopped**: For services you want to be automatically restarted but can be manually stopped when required.
* **on-failure**: For services where you expect occasional failures (e.g., batch jobs or unreliable network services).
* **no**: For containers where you want full control and don't need automatic restarts.

**docker run -d \**

**--name my-container \**

**--network my-network \**

**-v /path/to/host/volume:/path/in/container \**

**-v /another/host/volume:/another/path/in/container \**

**my-image**

### ****Docker Networking Questions:****

1. **What is Docker networking?**
   * Explain the concept of Docker networking, which allows containers to communicate with each other and external systems.
2. **What are the different types of Docker networks?**
   * Discuss the types of Docker networks: **bridge**, **host**, **none**, and **custom networks** (e.g., overlay and macvlan).
3. **What is the default Docker network when no network is specified?**
   * The default network for containers when no explicit network is defined is the **bridge** network.
4. **What is a Docker bridge network?**
   * Explain the bridge network, the default network mode for containers, which creates an internal network and attaches containers to it.
5. **How do you connect a container to a Docker network?**
   * Containers are connected using the --network flag when running a container: docker run --network <network-name> <image>.
6. **What is the purpose of the docker network ls command?**
   * The docker network ls command lists all the networks available in Docker.
7. **What is the difference between a host network and a bridge network in Docker?**
   * The **host** network mode connects the container directly to the host’s network stack, whereas the **bridge** network mode isolates the container’s network.
8. **What is an overlay network in Docker?**
   * The overlay network enables containers running on different Docker hosts (in a swarm) to communicate with each other as if they were on the same network.
9. **What are Docker network aliases?**
   * Docker network aliases are alternate names given to containers on a custom network, allowing containers to communicate with each other by these names.
10. **How do you expose a port on a Docker container to the outside world?**
    * Use the -p flag to expose and map ports: docker run -p 8080:80 <image>.

### ****Docker Volumes Questions:****

1. **What is a Docker volume?**
   * A Docker volume is a persistent data storage mechanism used to store and share data between containers, providing data persistence even after container deletion.
2. **What is the difference between Docker volumes and bind mounts?**
   * Volumes are managed by Docker, provide more abstraction, and can be shared between multiple containers. Bind mounts, on the other hand, directly link a host directory to a container directory.
3. **How do you create a Docker volume?**
   * A volume can be created using the command: docker volume create <volume-name>.
4. **How do you mount a volume into a running container?**
   * Use the -v flag to mount a volume: docker run -v <volume-name>:/path/in/container <image>.
5. **What is the docker volume inspect command used for?**
   * The docker volume inspect command retrieves detailed information about a specific Docker volume, such as its mount point and usage.
6. **What happens to Docker volumes when a container is removed?**
   * Volumes persist even if the container is removed, unless the volume is explicitly deleted using docker volume rm.
7. **What is the difference between a Docker volume and a Docker bind mount in terms of storage location?**
   * Volumes are stored in Docker’s internal storage (usually under /var/lib/docker/volumes/), while bind mounts use a specific directory from the host filesystem.
8. **Can you share a volume between multiple containers?**
   * Yes, you can share a volume between multiple containers, and changes to the volume from one container will be reflected in the other.
9. **What are named and anonymous volumes in Docker?**
   * Named volumes are explicitly created with a name (e.g., docker volume create my\_volume), while anonymous volumes are created automatically when the -v flag is used without specifying a volume name.
10. **How do you remove a Docker volume?**
    * Use docker volume rm <volume-name> to remove a volume. Make sure the volume is not in use by any container.

These questions test a candidate's knowledge of Docker’s networking and storage concepts, which are crucial for managing containerized applications. Let me know if you need answers or more details for any specific question! 😊

4o mini

##### You said:

add this also why we need to create user define network

###### ChatGPT said:

### ****Docker Network Questions (Including User-defined Networks):****

1. **Why do we need to create a user-defined network in Docker?**

* **Explanation**: User-defined networks in Docker offer several advantages over the default bridge network:
  + **Custom DNS**: User-defined networks allow containers to resolve each other by container names using Docker's internal DNS.
  + **Isolated Network**: Containers on a user-defined network can communicate with each other, but they are isolated from other containers on different networks. This enhances security and networking flexibility.
  + **Better Control Over IP Addressing**: With user-defined networks, you can control IP address ranges and subnets for containers.
  + **Improved Communication Between Containers**: In a user-defined bridge network, containers can communicate with each other by name, making it easier to manage multi-container applications.

**Use Case**: When setting up complex applications, like microservices, where you want containers to interact securely and efficiently, creating a user-defined network gives more control over their communication and behavior.

1. **Using ip command**:

ip addr show

1. **Using ifconfig command**:

ifconfig

1. **Using hostname command**:

hostname -I

1. **Using nmcli command** (for NetworkManager-managed systems):

nmcli device show

1. **Using curl to check public IP**:

curl ifconfig.me

1. **ls** – List files and directories in the current directory.

ls

1. **cd** – Change the current directory.

cd /path/to/directory

1. **pwd** – Print the current working directory.

pwd

1. **mkdir** – Create a new directory.

mkdir directory\_name

1. **rmdir** – Remove an empty directory.

rmdir directory\_name

1. **rm** – Remove files or directories.

rm file\_name

1. **cp** – Copy files or directories.

cp source\_file destination

1. **mv** – Move or rename files or directories.

mv source\_file destination

1. **cat** – Concatenate and display the contents of a file.

cat file\_name

1. **nano** – Edit a file using the nano text editor.

nano file\_name

1. **vim** – Edit a file using the Vim text editor.

vim file\_name

1. **echo** – Print text to the terminal.

echo "Hello, World!"

1. **top** – Display running processes and resource usage.

top

1. **ps** – Display information about running processes.

ps aux

1. **df** – Display disk space usage.

df -h

1. **du** – Display disk usage of files and directories.

du -sh directory\_name

1. **chmod** – Change file permissions.

chmod 755 file\_name

1. **chown** – Change file owner and group.

chown user:group file\_name

1. **grep** – Search for text within files.

grep "search\_string" file\_name

1. **man** – View the manual for a command.

man ls