**What Volume Mounting ?**

\*\*Volume Mounting\*\* refers to the process of making a directory or file on your host machine (your local computer) accessible within a Docker container. This is done by linking a directory or file from your host system to a directory or file inside the container.

**\*\*Key Concepts of Volume Mounting\*\*:**

1. \*\*Host Directory to Container Directory\*\*:

- When you mount a volume, you specify a directory on your host machine that you want to share with a directory inside the container.

- This allows files and data to be shared between the host and the container. Changes made in the mounted directory on the host machine will be reflected inside the container, and vice versa.

2. \*\*Two Types of Mounts\*\*:

- \*\*Bind Mounts\*\*: These are specific paths on your host machine that are mounted into the container. For example, `-v /path/on/host:/path/in/container`.

- \*\*Named Volumes\*\*: These are Docker-managed volumes that are stored in a specific location on your host machine, but you don’t usually specify the exact path. Docker manages these volumes, and they are typically used for data persistence.

3. \*\*Why Volume Mounting is Useful\*\*:

- \*\*Data Persistence\*\*: Volumes allow data to persist beyond the lifecycle of a container. When a container is deleted, the data in a volume can still be accessed.

- \*\*Development\*\*: In development, volume mounting is used to sync code between your host machine and the container, so you can see changes live without rebuilding the image.

- \*\*Configuration Sharing\*\*: Volumes can be used to share configuration files or other data between the host and container.

### \*\*Example of Volume Mounting\*\*:

```bash

docker run -v /path/on/host:/path/in/container my-image

```

-v /path/on/host:/path/in/container:

- \*\*`/path/on/host`\*\*: This is a directory on your host machine.

- \*\*`/path/in/container`\*\*: This is the directory inside the container where the host directory will be mounted.

- The container will see the contents of `/path/on/host` inside `/path/in/container`, and any changes will be synchronized.

\*\*Use Case Example\*\*:

- \*\*Web Development\*\*:

- You’re working on a website project stored at `/myproject` on your host machine.

- You run a Docker container for your web server, and you want the server to use your local files directly.

- You use volume mounting: `docker run -v /myproject:/var/www/html my-web-server`.

- Now, any changes you make to the files in `/myproject` on your host are immediately reflected inside the container at `/var/www/html`, and the web server will serve the latest content without needing to rebuild the Docker image.

Now as you have experience that like we have created one container , now let suppose I am continuously making changes in app, ab agar un changes ko container may reflect krna hai toh its mean kay dubara say ek Docker image build krni hogi, Now agar ek do dafa change krna ho app toh smjh ata hai bnda dubara say image create krlay , But agar frequent changes ho rhay hain app may and you to automate this process kay jo changes maa local system may app pay krunga wo automatically reflect hotay jayen container may be .

So to achieve this purpose we can use **Docker Compose** as well which is latest tool, but right now we will learning how to automate this process manually.

\*\*Command\*\*:

**Docker run -p 5173:5173 -v “${pwd):/app” -v /app/node\_modules react-docker**

\*\*1. `docker run`\*\*:

- \*\*What It Does\*\*: Starts a new container from the `react-docker` image.

\*\*2. `-p 5173:5173`\*\*:

- \*\*Port Mapping\*\*: This maps port `5173` on your host machine (your computer) to port `5173` inside the Docker container.

- \*\*Purpose\*\*: This allows you to access the application running inside the container on port `5173` by going to `http://localhost:5173` on your host machine.

\*\*3. `-v "$(pwd):/app"`\*\*:

- \*\*Volume Mounting (First One)\*\*:

- \*\*`-v "$(pwd):/app"`\*\*: This mounts the current directory on your host machine (where you run the command) to the `/app` directory inside the container.

- \*\*Purpose\*\*: This allows you to share your local files with the container. Any changes you make to the files on your host machine will be reflected inside the container, and vice versa.

\*\*4. `-v /app/node\_modules`\*\*:

- \*\*Volume Mounting (Second One)\*\*:

- \*\*`-v /app/node\_modules`\*\*: This is an anonymous volume that mounts the `node\_modules` directory inside the container.

- \*\*Purpose\*\*: It prevents the `node\_modules` directory from being overwritten by the empty directory in your host machine (since the previous `-v` command syncs the entire `/app` directory). This ensures that the container uses the `node\_modules` installed inside the container, not from your host machine.

\*\*5. `react-docker`\*\*:

- \*\*Image Name\*\*: This is the name of the Docker image you’re running. The container will start from this image.

### \*\*Summary\*\*:

Ab basic first we have mounted volume (it’s a process jokay humari jo host machine pay working directory hoti hai usko link krwata hai container kay andar jo WorkDir hoti hai) , Ab yaha be yehi hua kay when we put this command **-v “$(pwd): /app”** so basically it mounts our present working directory (pwd) into the **/app** WORKDIR in container. And now its allow kay jo be changes present working directory ma hongay on local machine wohi container mabi reflect kreingay. Or inkay darmayain jo syncing that is done by **Volume**. Or basically jo be changes hongay wo Volume may persist rahegay takay agar in future container crash hojaye toh jo Volume ma data hai usko krka **react-docker** image ka through new container ban jayega. This is also known as **Bind Mount.** Bcuz we are binding local machine file with container working directory through Volume.

Now what is the purpose creating this volume **-v /app/node\_modules** ?

its mean -v /app/node\_modules by doing this we have created another volume jismay saray node modules hain or yeh iss lia kia hai taka jo current working directory mount hogi from local machine to container toh usmay be ek node\_modules ki file hogi but wo empty hogi , toh iss lia kahin wo empty file overwrite naa krday node\_modules ki file ko jo container may hai. toh isi lia ek alag say volume create krliya container jisme /app/node\_moudles may jo node modules hain wo store krliya and yeh ensure krtay hain kay ab jo node\_modules directory hai inside container it will be protected now.

**\*\*Volume Mounting with `-v /app/node\_modules`\*\*:**

1. \*\*Mounting the Current Working Directory\*\*:

- When you use `-v "$(pwd):/app"`, you are mounting your local working directory (where your source code is) to the `/app` directory inside the container.

- This means that everything in your local directory, including a potentially empty `node\_modules` directory, will be mirrored inside the container.

2. \*\*The Problem\*\*:

- If your local directory contains an empty `node\_modules` folder, it could overwrite the `node\_modules` directory inside the container when the volume is mounted.

- This would be an issue because the container’s `node\_modules` directory contains all the installed dependencies required for your application to run.

3. \*\*Solution with `-v /app/node\_modules`\*\*:

- By using `-v /app/node\_modules`, you are creating a separate volume specifically for the `node\_modules` directory inside the container.

- This volume is independent of the local directory that is being mounted. It ensures that the `node\_modules` directory inside the container is preserved and not overwritten by the empty `node\_modules` directory from your local machine.

\*\*Summary\*\*:

- This ensures that the dependencies inside the container remain intact and are not accidentally removed when the local directory is mounted to `/app` inside the container.