🐳 Docker Overview

🔹 What is Docker?

Docker is a containerization platform that lets you package applications and their dependencies into lightweight containers that can run anywhere (your laptop, server, or cloud).

It eliminates the "works on my machine" problem.

🔹 Key Features of Docker

1. **Lightweight – uses host OS kernel (smaller than VMs).**
2. **Portability – build once, run anywhere (cloud, laptop, data center).**
3. **Isolation – each app runs in its own container.**
4. **Scalability – easily scale apps using Docker Compose / Swarm / Kubernetes.**
5. **Faster Deployment – apps start in seconds.**
6. **Image Versioning – store and share app images.**
7. **Security – controlled access, isolated environment.**
8. **Low cost compare to vm**

🔹 Use Cases

✅ **Microservices deployment (Node.js, Python, Java apps) / monolithic app**
✅ **DevOps CI/CD pipelines (Jenkins builds inside Docker)**
✅ **Cloud migrations (lift & shift apps in containers)**
✅ **Testing different environments (Python 3.8 vs 3.11 side-by-side)**
✅ **Databases & caching (MySQL, MongoDB, Redis in containers)**
✅ **Developer sandbox (no need to install heavy software locally)**

⚙️ Installing Docker

On Amazon Linux 2

```
sudo yum update -y

sudo amazon-linux-extras enable docker

sudo yum install -y docker

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker ec2-user

newgrp docker   or sudo reboot

docker --version
```

On Ubuntu / Debian

```
sudo apt-get update -y

sudo apt-get install -y ca-certificates curl gnupg lsb-release

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /etc/apt/keyrings/docker.gpg

echo \

  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg] \

  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
\
```

```
  | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update -y

sudo apt-get install -y docker-ce docker-ce-cli containerd.io

sudo systemctl start docker

sudo systemctl enable docker

docker --version
```

install and run Docker on AWS Linux (Amazon Linux) without rebooting.
Amazon Linux comes in **two major versions**:

- **Amazon Linux 2** (most commonly used for EC2 instances)
- **Amazon Linux 2023** (newer, `dnf` package manager)

Here are the **no-reboot steps** for both:

🔹 Amazon Linux 2

```
# 1. Update system
sudo yum update -y

# 2. Install Docker
sudo amazon-linux-extras enable docker
sudo yum install -y docker

# 3. Start Docker service immediately
sudo systemctl start docker
sudo systemctl enable docker
```

```
# 4. (Optional) Allow EC2 user to run Docker without sudo
sudo usermod -aG docker ec2-user

# 5. Verify Docker is running
docker --version
docker run hello-world
```

⚠️ If you add `ec2-user` to the `docker` group, you'd normally need to **log out and back in** (not reboot).
To avoid logout, just run:

```
newgrp docker
```

🔹 Amazon Linux 2023

```
# 1. Update system
sudo dnf update -y

# 2. Install Docker
sudo dnf install -y docker

# 3. Start Docker service immediately
sudo systemctl start docker
sudo systemctl enable docker

# 4. (Optional) Add ec2-user to docker group (no reboot needed)
sudo usermod -aG docker ec2-user
newgrp docker

# 5. Verify
docker --version
docker run hello-world
```

✅ This way your **Docker works right after install**, without reboot.

Do you want me to also give you a **single script** that you can copy-paste into a fresh AWS EC2 (Amazon Linux) instance to get Docker ready instantly?

You said:
how to allocate memory , volume , port mapping to container , cross check

ChatGPT said:

Perfect question 👍 You want to know how to **allocate resources (memory, volume, ports) to a Docker container** and then **cross-check** if it's applied.

Here's a clear step-by-step:

🔹 1. Allocate Memory & CPU

When running a container:

```
docker run -d \
  --name myapp \
  --memory="512m" \
  --cpus="1.0" \
  nginx
```

- `--memory="512m"` → limits container RAM usage to 512 MB
- `--cpus="1.0"` → allows container to use 1 CPU core

🔹 2. Allocate Volume (Persistent Storage)

```
docker run -d \
  --name myapp \
  -v /host/data:/container/data \
  nginx
```

- `-v /host/data:/container/data` → maps host folder `/host/data` to container's `/container/data`

🔹 3. Allocate Port Mapping

```
docker run -d \
  --name myapp \
  -p 8080:80 \
  nginx
```

- `-p 8080:80` → maps host port `8080` to container's port `80`
- Access app in browser: `http://<EC2-IP>:8080`

🔹 4. Combine All (Memory + Volume + Port)

```
docker run -d \
  --name myapp \
  --memory="512m" \
  --cpus="1.0" \
  -v /host/data:/container/data \
  -p 8080:80 \
  nginx
```

🔹 5. Cross-Check (Verify)

1. **Check container resource limits**

```
docker inspect myapp --format='{{.HostConfig.Memory}}
{{.HostConfig.NanoCpus}}'
```

(Memory is in bytes, NanoCPUs = 1e9 per CPU)

2. **Check port mapping**

```
docker port myapp
```

3. **Check volume mapping**

```
docker inspect myapp --format='{{json .Mounts}}' | jq
```

(or simply: `docker inspect myapp | grep Mounts -A 5`)

4. **Check container stats (live usage)**

```
docker stats myapp
```

⚡ Example:

- Start `nginx` with **512MB RAM, 1 CPU, port 8080, and volume mapping**
- Then verify with `docker inspect` and `docker stats`