What is Docker?

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization.

Key Concepts:

1. **Containers:**
    - Containers are lightweight, stand-alone, and executable software packages that include everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
    - Containers isolate the application from its environment, ensuring it works uniformly despite differences, for instance, between development and staging.
2. **Images:**
    - Docker images are read-only templates used to create containers.
    - Images include the application code, libraries, environment variables, configuration files, and other dependencies needed to run the application.
    - Docker images are built from Dockerfiles and can be stored in a Docker registry like Docker Hub.
3. **Dockerfile:**
    - A Dockerfile is a script containing a series of instructions on how to build a Docker image.
    - Common instructions include specifying the base image (`FROM`), running commands (`RUN`), copying files (`COPY`), and defining environment variables (`ENV`).
4. **Docker Hub:**
    - Docker Hub is a cloud-based registry service where Docker images are stored and shared.
    - Users can download pre-built images from Docker Hub or upload their own images.

How Docker Works:

1. **Docker Engine:**
    - The Docker Engine is the core component of Docker, responsible for creating and running containers.
    - It consists of a server (`dockerd`), a REST API for interacting with the server, and a command-line interface (`docker`).
2. **Container Lifecycle:**
    - **Build:** Create a Docker image from a Dockerfile.
    - **Run:** Create a container from a Docker image.
    - **Stop:** Stop a running container.
    - **Remove:** Delete a stopped container.
    - **Push:** Upload an image to a Docker registry.
    - **Pull:** Download an image from a Docker registry.

Benefits of Docker:

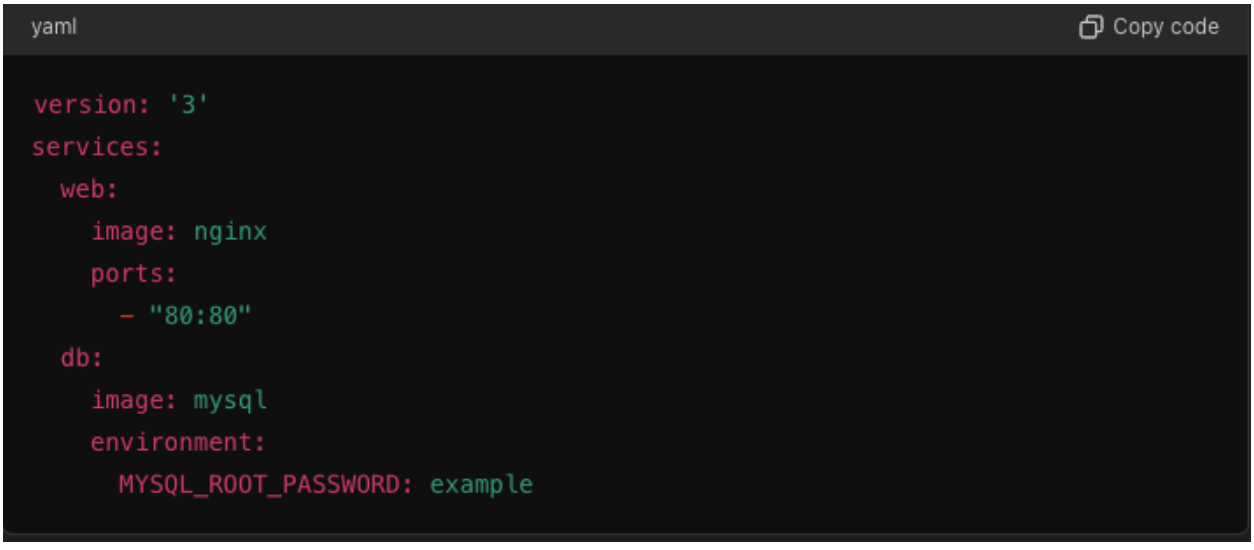
1. **Portability:**
   - Docker containers run consistently on any environment that supports Docker, eliminating the "it works on my machine" problem.
2. **Isolation:**
   - Containers run in isolation from each other, allowing multiple applications or services to run on a single host without conflicts.
3. **Efficiency:**
   - Containers are lightweight and share the host system's kernel, reducing overhead compared to virtual machines (VMs).
4. **Scalability:**
   - Docker makes it easy to scale applications horizontally by adding more containers as needed.

Docker Compose:

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the application's services, networks, and volumes.

Key Concepts:

1. **docker-compose.yml:**
   - The configuration file where you define your application's services, networks, and volumes.
   - Example:



2. **Commands:**
   - `docker-compose up`: Start all the services defined in `docker-compose.yml`.
   - `docker-compose down`: Stop and remove the containers, networks, and volumes created by `docker-compose up`.
   - `docker-compose logs`: View output from the services.

Docker Networking:

Docker Networking enables containers to communicate with each other and with the outside world.

Types of Docker Networks:

1. **Bridge Network:**
    - Default network type. Containers on the same bridge network can communicate with each other using IP addresses or container names.
2. **Host Network:**
    - Removes network isolation between the container and the Docker host. The container uses the host's network directly.
3. **Overlay Network:**
    - Enables containers to communicate across different Docker hosts, suitable for swarm services.
4. **Custom Network:**
    - Users can create custom networks to specify subnet configurations and other advanced network settings.

Docker Volumes:

Docker Volumes are used for persistent storage. They are managed by Docker and can be shared between containers.

Types of Volumes:

1. **Named Volumes:**
    - Created and managed by Docker. Data in named volumes persists even if the container is removed.
2. **Anonymous Volumes:**
    - Created when a container is started. If not specified, Docker assigns a unique name.
3. **Bind Mounts:**
    - Mounts a file or directory from the host machine into the container. Useful for development where code needs to be frequently updated.

Common Docker Commands:

1. **Build and Run Containers:**
    - `docker build -t my-image .`: Build a Docker image from a Dockerfile.
    - `docker run -d --name my-container my-image`: Run a container from an image in detached mode.

2. **Manage Containers:**
    - `docker ps`: List running containers.
    - `docker stop my-container`: Stop a running container.
    - `docker rm my-container`: Remove a stopped container.
3. **Manage Images:**
    - `docker images`: List all Docker images.
    - `docker rmi my-image`: Remove a Docker image.
4. **Networking:**
    - `docker network ls`: List all Docker networks.
    - `docker network create my-network`: Create a new Docker network.
5. **Volumes:**
    - `docker volume ls`: List all Docker volumes.
    - `docker volume create my-volume`: Create a new Docker volume.