DOCKER AND CONTAINERIZATION STUDY PLAN

WEEK 1: FUNDAMENTALS AND HISTORY OF DOCKER AND CONTAINERIZATION

Goal: Understand the problem that containers solve and how they provide the solution.

Topics:

- **What is containerization?** Learn the concept of application bundling, such as its dependencies, into a single, isolated package.
- **Containers versus Virtual Machines:** Understand the key differences, performance benefits, and use cases for each. Such as how containers virtualize the operating system, while virtual machines virtualize the hardware.
- **History of Docker:** Explore the evolution of early concepts like chroot in 1979 and FreeBSD Jails to Linux Containers (LXC) and the rise of Docker in 2013.
- **The Docker Ecosystem:** Get a high-level overview of the Docker platform, including its engine, hub, and desktop application.

Tasks:

- Read the "What is a Container?" document on the official Docker website.
- Watch a video lesson that explains the difference between containers and virtual machines.
- Install **Docker Desktop** on your computer.

WEEK 2: HANDS-ON ACTIVITIES

Goal: Achieve hands-on experience on how to use Docker.

Topics:

- **Docker CLI:** Get comfortable with the basic structure of Docker commands. (similar to Git CLI).
- **Key Commands:**
  1. **docker run –** run the container from an image (a package full of things needed to run a container)
  2. **docker ps** – lists running containers (they can be of different OS, etc.)
  3. **docker images** – list images you have locally.
  4. **docker pull** – download an image from a registry like Docker Hub.
  5. **docker rm / docker rmi** – used to remove containers and images.

Tasks:

- Open terminal and run first container.
- Pull and run a more complex image. (e.g a webserver, a different OS, etc.)
- Practice listing, stopping, and removing specific containers and images.

DOCKER AND CONTAINERIZATION STUDY PLAN

WEEK 3: BUILDING YOUR OWN DOCKERFILE

Goal: Be able to create custom container images.

Topics:

- **What is a DockerFile?** Learn what a DockerFIle is and how it is used to build a Docker Image.
- **Common Instructions:** Learn the purpose of FROM, WORKDIR, COPY, RUN, AND CM.
- **Image Layers:** Understand how each instruction in a DockerFile creates a new layer, and why this is efficient.
- **How the .dockerignore file works**: Learn how to exclude files from build context to keep images small and secure (Similar to how .gitignore works).

Tasks:

- Create simple “Hello World” application, in any language you want.
- Write a DockerFile for said application.
- Build first custom image using the Docker CLI.
- Run a container from the newly built image and verify it works.

WEEK 4: DATA AND NETWORKING IN DOCKER

Goal: Learn how containers manage persistent data and communicate with each other.

Topics:

- **Docker Volumes:** The preferred way to persist data generated by and used by Docker containers, and how data in a volume exists outside the container’s lifecycle.
- **Bind Mounts:** An alternative way to share a directory from your host machine directly into a container.
- **Docker Networking:** Understand the default bridge network that allows containers on the same host to communicate.
- **Port Mapping:** Review how to expose a container's port to the host machine's network (-p flag).

Tasks:

- Create a container that uses a volume to store data (e.g., a database like Postgres).
- Stop and remove the container, then create a new one attached to the *same volume* and see that your data is still there.
- Create a custom bridge network (docker network create my-net) and run two containers on it. Practice making them communicate with each other by name.

DOCKER AND CONTAINERIZATION STUDY PLAN

WEEK 5: MULTI-CONTAINER APPS WITH DOCKER COMPOSE

Goal: Understand how to use multiple containers in applications for multiple services.

Topics:

- **What is Docker Compose?** Learn how to use Docker Compose for defining and running multi-container Docker applications using a simple YAML file (docker-compose.yml).
- **AML File Structure:** Learn how to define services, networks, and volumes in a docker-compose.yml file.
- **Key Commands:** Use docker-compose up to start your application stack and docker-compose down to stop and remove it.

Tasks:

- Create a docker-compose.yml file for a simple application with two services (e.g., a web application and a database).
- Use docker-compose up to launch your application.
- Practice scaling a service up or down using the --scale flag.

WEEK 6: BEST PRACTICES AND ADVANCED STEPS

Goal: Solidify knowledge by implementing Docker professionally and efficiently.

Topics:

- **Dockerfile Best Practices:** Learn how to implement multi-stage builds, using specific image tags (not :latest), minimizing layers, and running as a non-root user.
- **Security:** Apply basic container security principles, such as scanning images for vulnerabilities.
- **Image Optimization:** Keeping your Docker images as small as possible for faster builds and deployments.
- **Introduction to Orchestration:** Briefly learn what tools like **Kubernetes** and **Docker Swarm** are and why they are used to manage containers at scale.

Tasks:

- Refactor one of your previous Docker files to use a multi-stage build.
- Research and find 3 common Docker security vulnerabilities.
- Sign up for a free Docker Hub account and push one of your custom images to it.

DOCKER AND CONTAINERIZATION STUDY PLAN

BEST PRACTICES IN IMPLEMENTING DOCKER

- **Keep Images Small:** Use smallest possible images such as alpine or slim variants and utilize .dockerignore to reduce layers.
- **Use Multi-Stage Builds:** Use one stage to build application artifact and a separate, smaller stage with only necessary runtime to run it.
- **Specify Image Tags:** Avoid using the :latest tag in your FROM instruction, and specify tags to versions.
- **Run as a Non-Root User:** Avoid running containers as the root user to limit potential damage to compromised containers.
- **Use Build Cache:** Structure your DockerFIle to utilize layer caching, and place instructions that change frequently at the end of the file if possible.
- **Use Official Images:** Stick to using official images from the Docker Hub, as they are regularly scanned for vulnerabilities and maintained by fellow developers.
- **Practice Security:** Never hard-code secrets like API keys or passwords in the DockerFile or image, and utilize environment variables or Docker's built-in secrets management.
- **Utilize Logging:** Use STDOUT and STDERR to log standard output and error, allowing Docker to collect and manage logs effectively.