Assignment 1

Docker Assignment

This assignment will test your understanding of basic Docker concepts and commands. You will be working with Docker images, containers, and Docker Compose. This assignment has four parts. Please submit the answers for the assignment in a single PDF file called Assignment1.pdf.

Part A: Docker Command Explanations (20 Points)

Explain the purpose and usage of the following Docker commands. Provide examples where appropriate.

1. `docker build` **(10 Points)**
    - What does this command do?
    - What is the primary purpose of a `dockerfile` in the context of `docker build`?
    - Explain the `-t` option used with `docker build`.

2. `docker run` **(10 Points)**
    - What does this command do?
    - Explain the following options commonly used with `docker run`:
        - `-p` (port mapping)
        - `-v` (volume mounting)
        - `-it` (interactive and pseudo-TTY)
        - `--name` (container naming)

Part B: Creating a Python Docker Image (30 Points)

Create a `dockerfile` that builds a Docker image with the following specifications:

1. **Base Image:** Use the most recent official Python image available on Docker Hub. (Hint: Use the `python:latest` tag).
2. **Working Directory:** Set the working directory inside the container to `/app`.

3. **Exposed Port:** Expose port `8080` from the container.
4. **Default Command:** Set the default command to `bash` so that when a container is started from this image, you get an interactive command-line shell.
5. **Build and Tag:** After writing the `dockerfile`, use the `docker build` command to build the image. Tag the image as `my/python`.

Deliverables for Part B:

- The content of your `dockerfile`.
- The command you used to build the image.

Part C: Running a Container with Specific Configurations (20 Points)

Write the `docker run` command that starts a container with the following characteristics:

1. **Image:** Use the `my/python` image you created in Part B.
2. **Container Name:** Name the container `python1`.
3. **Port Mapping:** Map port `8080` on your host machine to port `8080` inside the container.
4. **Volume Mounting:** Mount a directory from your host machine to the `/app` directory inside the container. You can choose any directory on your host machine (e.g., `/Users/yourname/demo/python` on macOS, `C:\Users\yourname\demo\python` on Windows, or `/home/yourname/demo/python` on Linux).
5. **Interactive Shell:** Ensure you have an interactive shell (command line) when the container starts.

Deliverables for Part C:

- The complete `docker run` command you used.

Part D: Simplifying Container Management with Docker Compose (30 Points)

Create a `docker-compose.yml` file that achieves the same configuration as the `docker run` command you wrote in Part C. This means the Docker Compose file should:

1. **Define a Service:** Create a service (you can name it `python-app` or something similar).
2. **Specify Image:** Use the `my/python` image.
3. **Set Container Name:** Name the container `python1`.
4. **Configure Port Mapping:** Map port `8080` on the host to port `8080` in the container.
5. **Configure Volume Mounting:** Mount the same host directory you chose in Part C to `/app` inside the container.

6. **Enable Interactive Mode:** Ensure that the container starts in a way that allows you to interact with it through a shell (similar to using `-it` with `docker run`).

Deliverables for Part D:

- The content of your `docker-compose.yml` file.
- The command you use to start the container using Docker Compose.