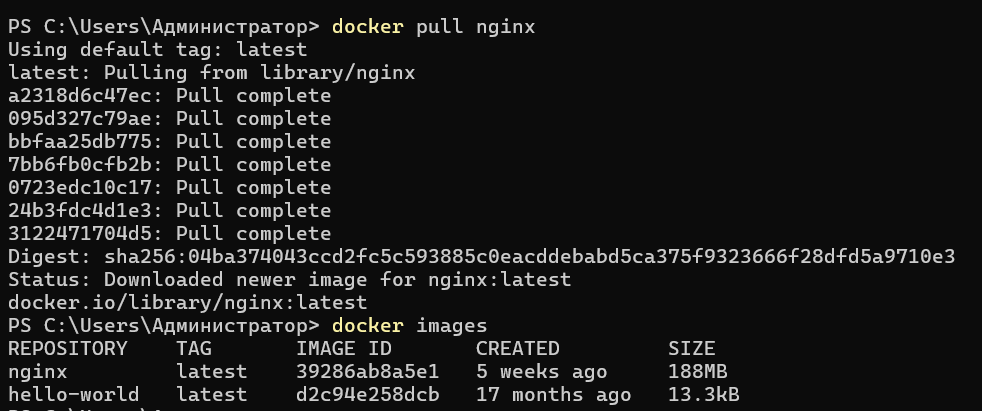
### **Intro to Containerization: Docker**

Exercise 1.

3. Questions:  
1) What are the key components of Docker (e.g., Docker Engine, Docker CLI)?  
Docker Images, Docker Containers, Docker Hub, Docker Compose, Docker Swarm, Docker Volumes  
2) How does Docker compare to traditional virtual machines?  
In Docker, applications run in containers. Containers are quick, lightweight and share the host OS Kernel. Virtual Machine uses its own OS each, so it uses more resources compared to Docker. From that we can say that containers have less memory and storage, start quickly, and are highly portable across different environments. For cons, Dockers are less secure than VMs, because all containers share the same OS.

Exercise 2.



1. What is the difference between docker pull and docker run?  
   Docker pull - downloades the image from a registry to the local machine. Docker run - starts a container from a specified image.
2. How do you find the details of a running container, such as its ID and status?  
   with the command docker ps.
3. What happens to a container after it is stopped? Can it be restarted?  
   its state is saved, and it can be restarted.
4. What was the output of the docker run hello-world command, and what does it signify?  
   It signifies that Docker is running properly and it can be used to run the applications.

Exercise 3.  
1. How does port mapping work in Docker, and why is it important?  
It makes services running in containers accessible from outside. It makes development flexible, deployment of multiple services, maintains the isolation.

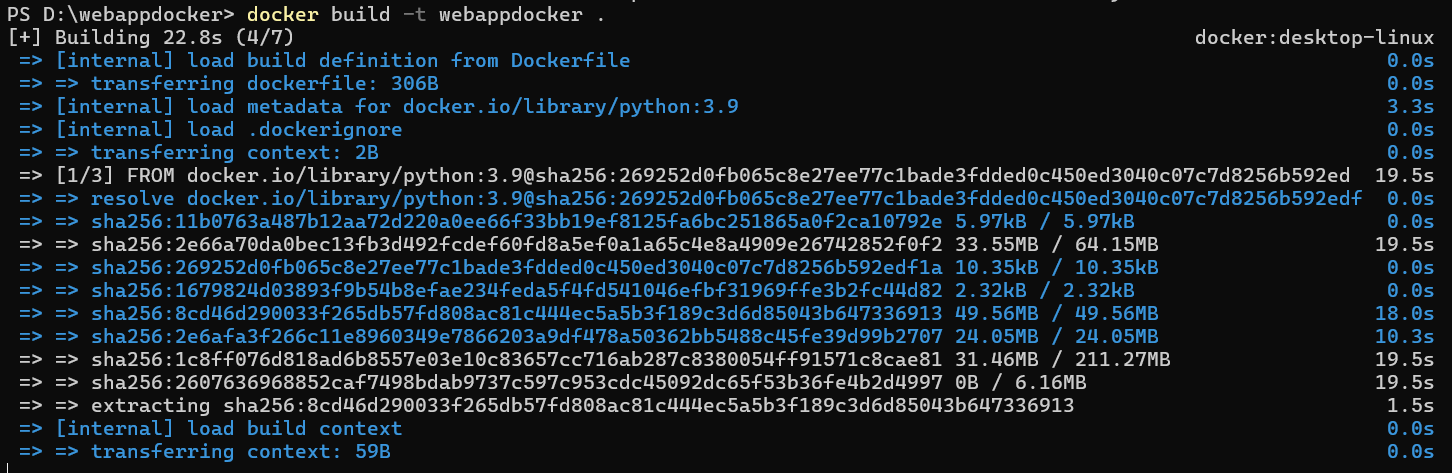
2. What is the purpose of the docker exec command?

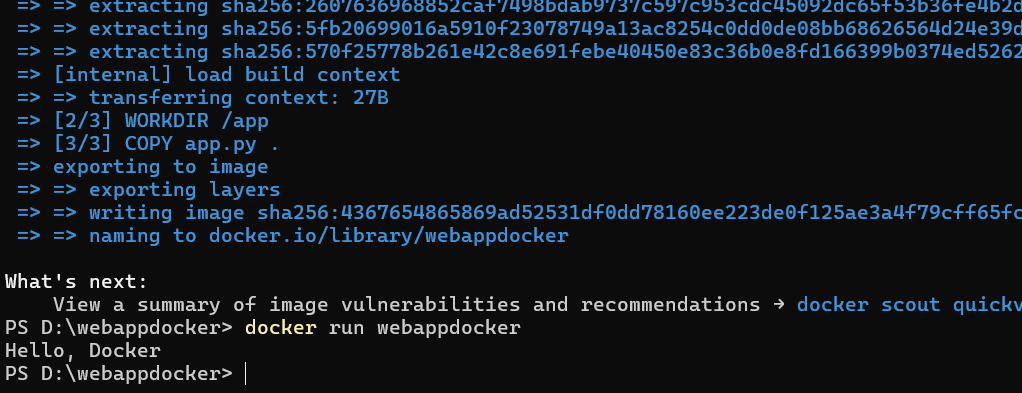
We access the shell of the container  
3. How do you ensure that a stopped container does not consume system resources?  
We use docker rm to remove the stopped container

### **Dockerfile**

#### **Exercise 1: Creating a Simple Dockerfile**

1. **Objective**: Write a Dockerfile to containerize a basic application.
2. **Steps**:
   * Create a new directory for your project and navigate into it.
   * Create a simple Python script (e.g., app.py) that prints "Hello, Docker!" to the console.
   * Write a Dockerfile that:
     + Uses the official Python image as the base image.
     + Copies app.py into the container.
     + Sets app.py as the entry point for the container.
   * Build the Docker image using docker build -t hello-docker ..
   * Run the container using docker run hello-docker.





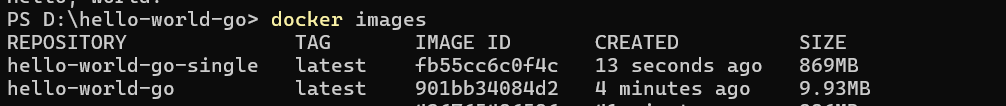
1. **Questions**:
   * What is the purpose of the FROM instruction in a Dockerfile?  
     specify the image
   * How does the COPY instruction work in Dockerfile?  
     copies app.py into the container
   * What is the difference between CMD and ENTRYPOINT in Dockerfile  
     entrypoint - building an executable Docker image, cmd - when you need an additional set of arguments

#### **Exercise 2: Optimizing Dockerfile with Layers and Caching**

1. **Objective**: Learn how to optimize a Dockerfile for smaller image sizes and faster builds.
2. **Steps**:
   * Modify the Dockerfile created in the previous exercise to:
     + Separate the installation of Python dependencies (if any) from the copying of application code.
     + Use a .dockerignore file to exclude unnecessary files from the image.
   * Rebuild the Docker image and observe the build process to understand how caching works.  
     Build was much quicker
   * Compare the size of the optimized image with the original.
3. **Questions**:
   * What are Docker layers, and how do they affect image size and build times?  
     Layers are main components of Docker images that enable efficient storage, sharing, and versioning of software. by efficiently managing them, it leads to smaller images and faster builds.
   * How does Docker's build cache work, and how can it speed up the build process?  
     it stores the intermediate layers.when rebuilding an image docker uses the cached layer instead of rebuilding. so it speeds up the process as it skips some steps
   * What is the role of the .dockerignore file?  
     allows patterns and wildcards to specify which files and directories to ignore.

#### **Exercise 3: Multi-Stage Builds**

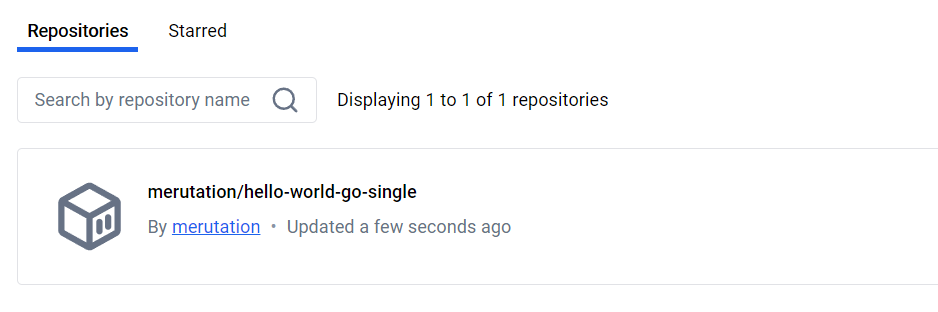
1. **Objective**: Use multi-stage builds to create leaner Docker images.
2. **Steps**:
   * Create a new project that involves compiling a simple Go application (e.g., a "Hello, World!" program).
   * Write a Dockerfile that uses multi-stage builds:
     + The first stage should use a Golang image to compile the application.
     + The second stage should use a minimal base image (e.g., alpine) to run the compiled application.
   * Build and run the Docker image, and compare the size of the final image with a single-stage build.



1. **Questions**:
   * What are the benefits of using multi-stage builds in Docker?  
     prevents from unnecessary build tools and dependencies, minimizes potential vulnerabilities, cleaner dockerfiles, faster builds
   * How can multi-stage builds help reduce the size of Docker images?  
     we can choose to copy only specific files or directories from one stage to another. This means you can exclude unnecessary files that are not needed in the final image. By organizing the build process into distinct stages, you can minimize the number of layers in the final image, which can help reduce overhead and size.
   * What are some scenarios where multi-stage builds are particularly useful?  
     When building applications with significant dependencies (e.g., Java, Go), you can compile and package the application in one stage, then copy only the necessary artifacts to a smaller runtime image. If you need to build different versions of an application (like debug and release versions), you can create different stages for each version, simplifying the process.

#### **Exercise 4: Pushing Docker Images to Docker Hub**

1. **Objective**: Learn how to share Docker images by pushing them to Docker Hub.
2. **Steps**:
   * Create an account on Docker Hub.
   * Tag the Docker image you built earlier with your Docker Hub username (e.g., docker tag hello-docker <your-username>/hello-docker).
   * Log in to Docker Hub using docker login.
   * Push the image to Docker Hub using docker push <your-username>/hello-docker.
   * Verify that the image is available on Docker Hub and share it with others.



1. **Questions**:
   * What is the purpose of Docker Hub in containerization?  
     for sharing and managing Docker images, promoting collaboration and efficiency in containerization.
   * How do you tag a Docker image for pushing to a remote repository?  
     docker tag hello-docker <your-username>/hello-docker
   * What steps are involved in pushing an image to Docker Hub?  
     build the image, tag it, push and verify the push