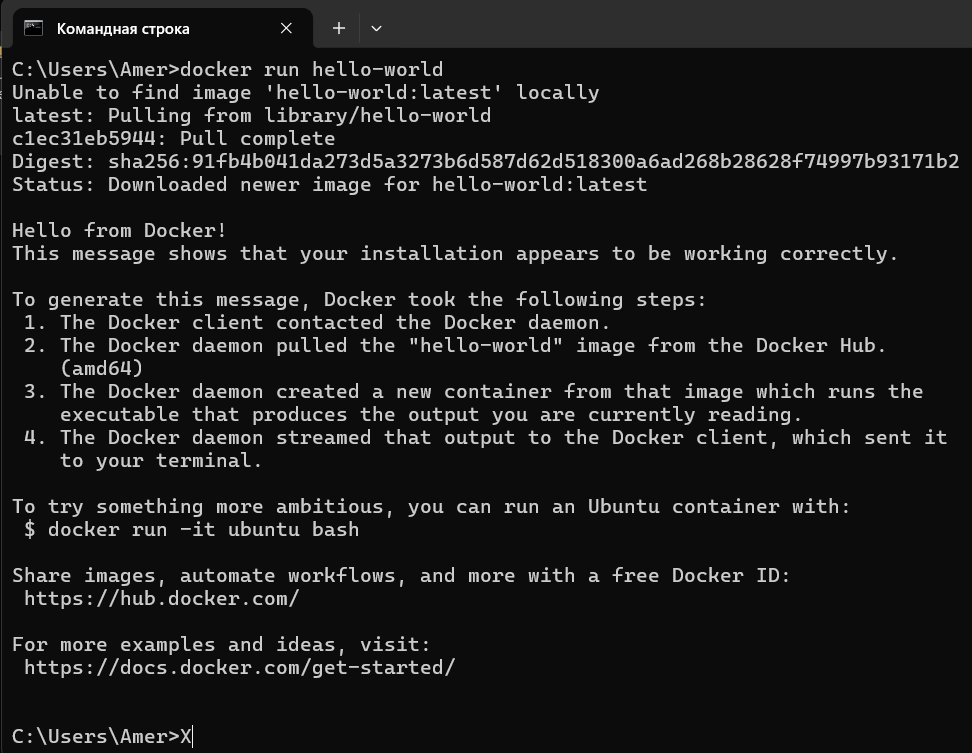
**Intro to Containerization: Docker**

**Exercise 1: Installing Docker**

1. Major components of the Docker architecture are: The Docker Daemon (Server), REST API (Docker Engine), Docker CLI (Client).
2. Virtual Machines (VM) run a full operating system (OS) inside another host OS. VMs are slow and require a lot of memory and CPU. Docker has a container; they share host OS. Containers are isolated processed in user space. Ther include only the application and its dependencies, making them much lighter than VMs.

VMs are less portable because they include the entire OS. Docker containers can run on any system that support Docker.

1. docker run hello-world shows message that your installation appears to be working correctly, Docker pulls the “hello-world” image from Docker Hub and runs it in container.

Output:  


**Exercise 2: Basic Docker Commands**

1. “docker pull” only download the image, “docker run” downloads the image (of not already available) and starts a container from it.
2. “docker ps” or “docker ps -a”. First one shows all running containers, second one shows all containers that are running and stopped.
3. Status will be “Exited”. Any data written to the container’s writable layer remains intact. The container releases the resources (CPU, memory) it was using.

Container can be restarted using “docker start <container-id or container name>”

**A screenshot of a computer program

Description automatically generated**

**Exercise 3: Working with Docker Containers**

1. By default, Docker containers are isolated from the host OS and other containers. They do not expose any ports to the outside world unless explicitly configured.

“docker run -d -p 8080:80 nginx.” Here -p to map a port on the host to a port in the container. Port mapping allows services running inside container to be accessible from the host machine or other external systems.

1. “docker exec” Executes a command in a running container.
2. “docker rm” removes the stopped container, freeing up the disk space it was using.

A screenshot of a computer

Description automatically generated

**Dockerfile**

**Exercise 1: Creating a Simple Dockerfile**

1. FROM: Specifies the base image to build upon.
2. COPY: Copies files from the host into the container.
3. “CMD” specifies the default command to run when a container is started. “ENTRYPOINT” defines the main command to run within the container. “CMD” can be easily overridden by specifying a different command at runtime, “ENTRYPOINT” is more rigid.

A screenshot of a computer

Description automatically generated

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

1. Docker Layers are set of filesystem changes, such as adding or modifying files. Creation layer, read only layer, caching. Minimize image size and reduce build times. But in my case optimized image became heavier to 10 MB(.
2. When we rebuild an image, Docker checks if it can reuse any of the cached layers. If the instructions and its context haven’t changed, Docker reuses the cached layer instead of rebuilding. By reusing layers, Docker reduces the time required to build and image. Saves CPU, memory and network bandwidth, making the build process more efficient.
3. .dockerignore tells Docker which files and directories to exclude when building the image. Excluding unnecessary files helps reduce the image size and build context.

A screenshot of a computer

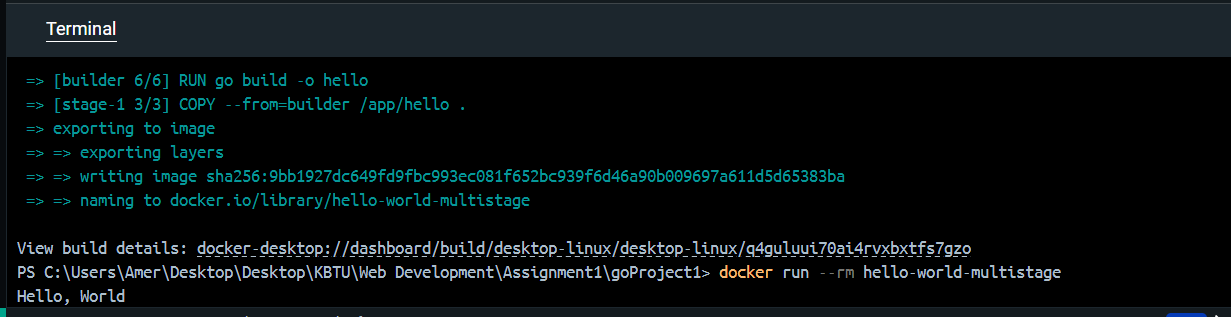
Description automatically generated

A screen shot of a computer

Description automatically generated

**Exercise 3: Multi-Stage Builds**

1. Allow to discard unnecessary build dependencies from the final image, resulting in a much smaller runtime image.
2. By using multi-stage build, you only copy over the final product into a minimal base image, leaving behind all unnecessary files, libraries, and compilers used during the build process.
3. Projects with a large dependency chains (Nodejs,Java) can use multi-stage builds to keep the final image free of build tools like npm, javac, etc.



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

1. It allows users to pull official, public, or private images and serves as a central hub for managing containerized applications.
2. “docker tag local-image-name username/repository-name:tag”
3. “docker login” then tag the image, then docker push username/repository-name:tag

A screenshot of a computer

Description automatically generated