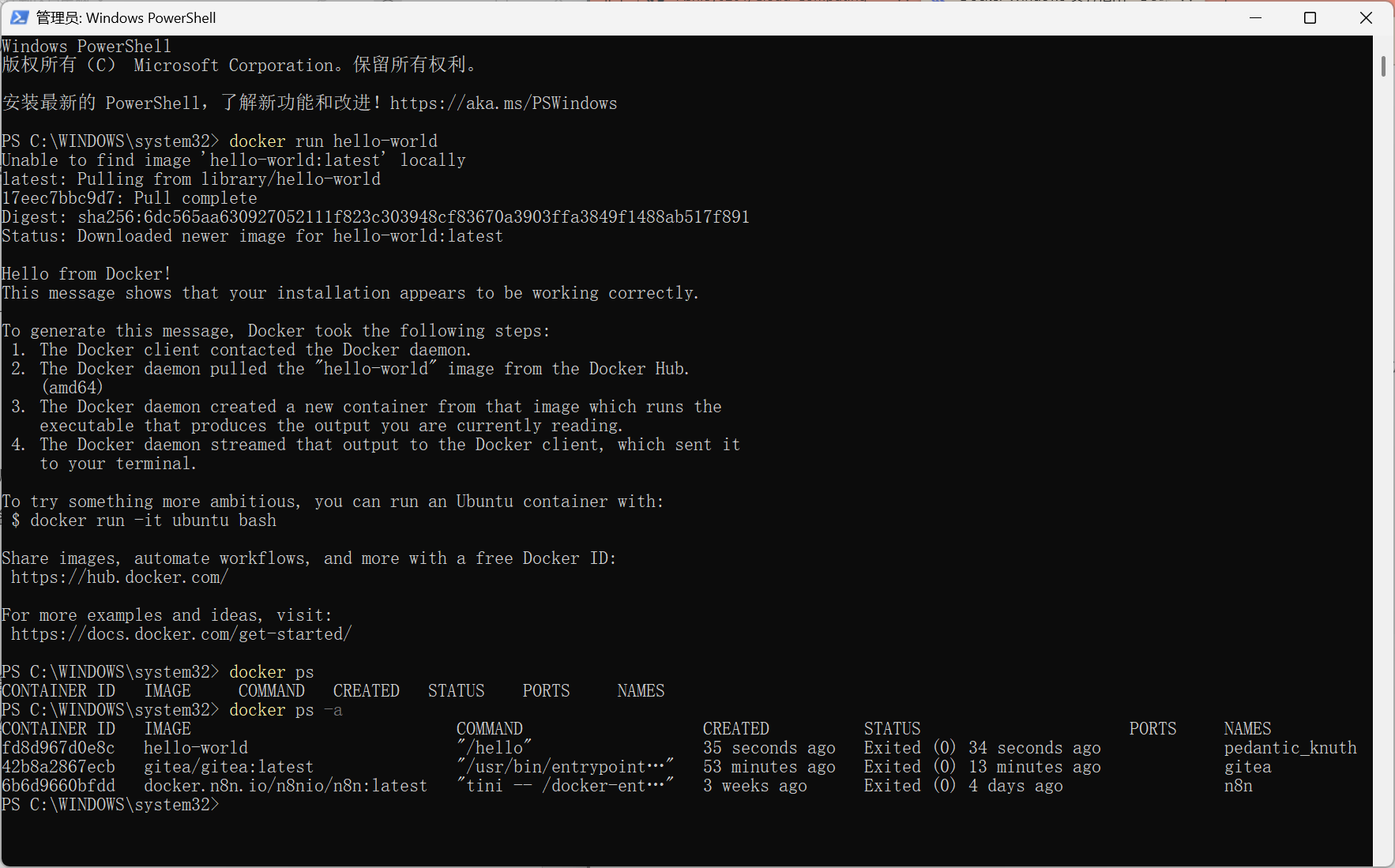
**1：Introduction to Docker(Container)**

**Container Concepts：**

* A container is a running instance of an image
* A lightweight, self-contained software package
* Contains everything needed to run an application

**Container Concepts：**

* Containers are temporary and data is lost if stopped (unless volumes are used)
* Each container has a unique ID and name

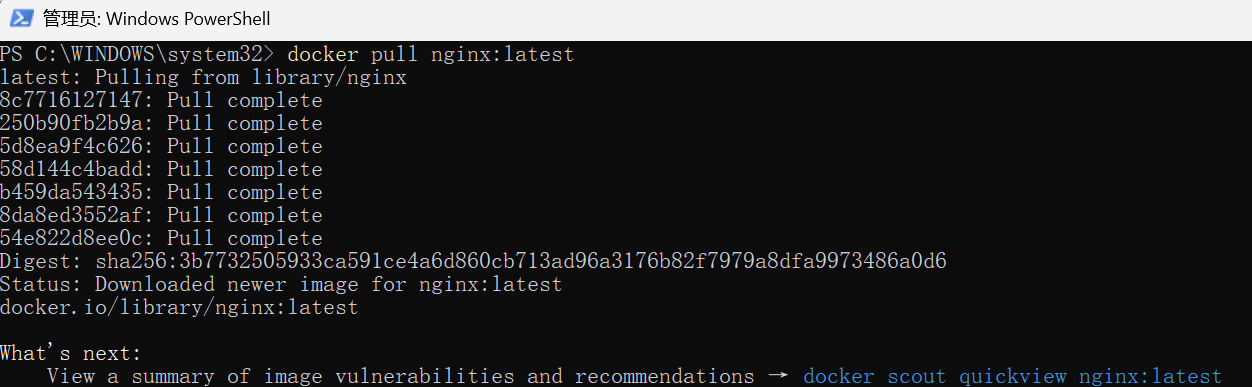
**2：Understanding Mirroring (Image)**

**Concepts：**

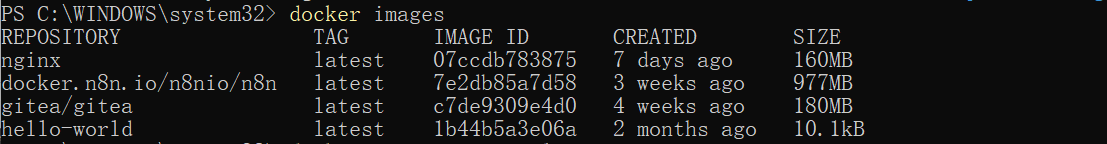
* An image is a read-only template that contains the instructions needed to create a container.
* Hierarchical storage structure improves reusability

**Experimental procedures：**

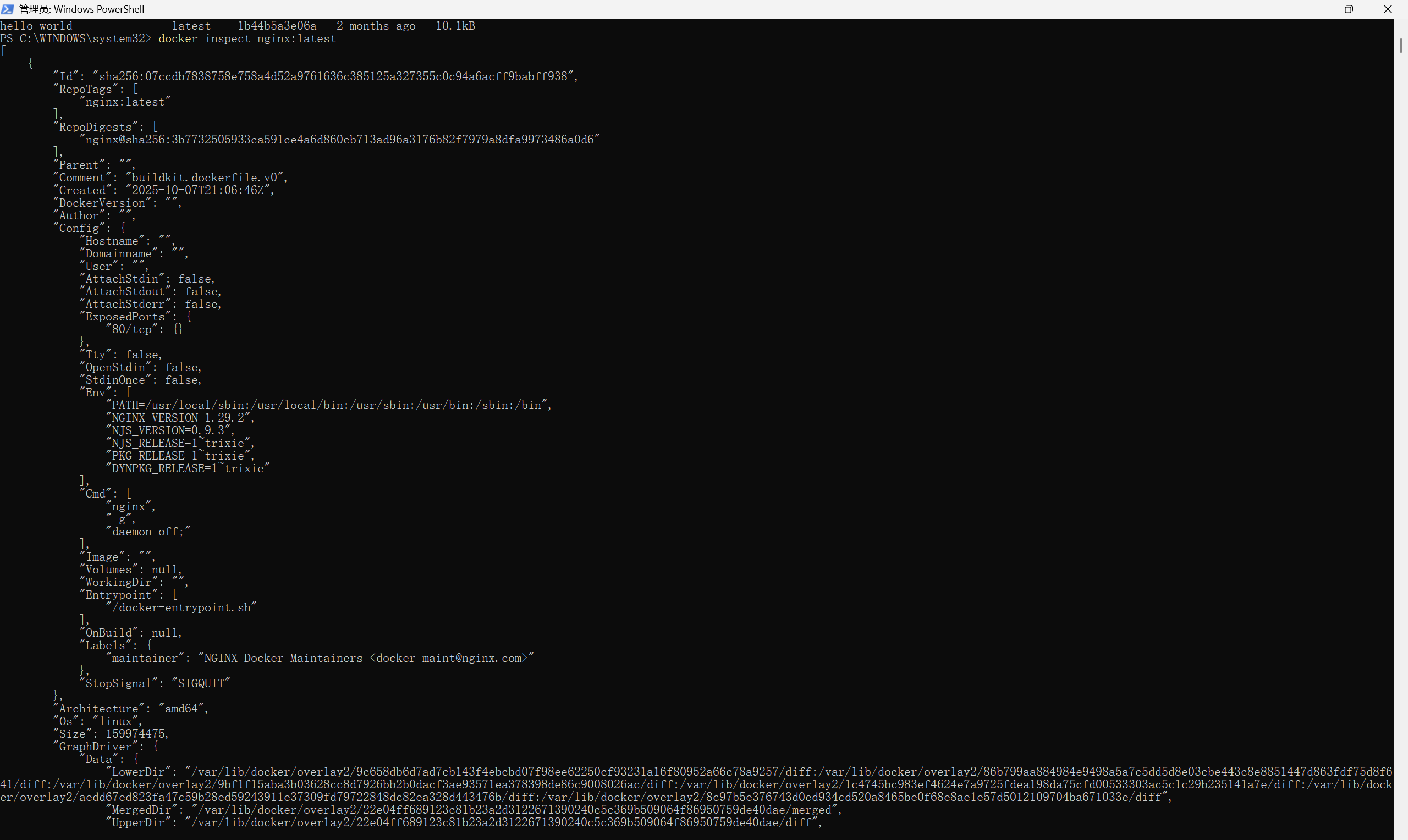
docker pull nginx:latest



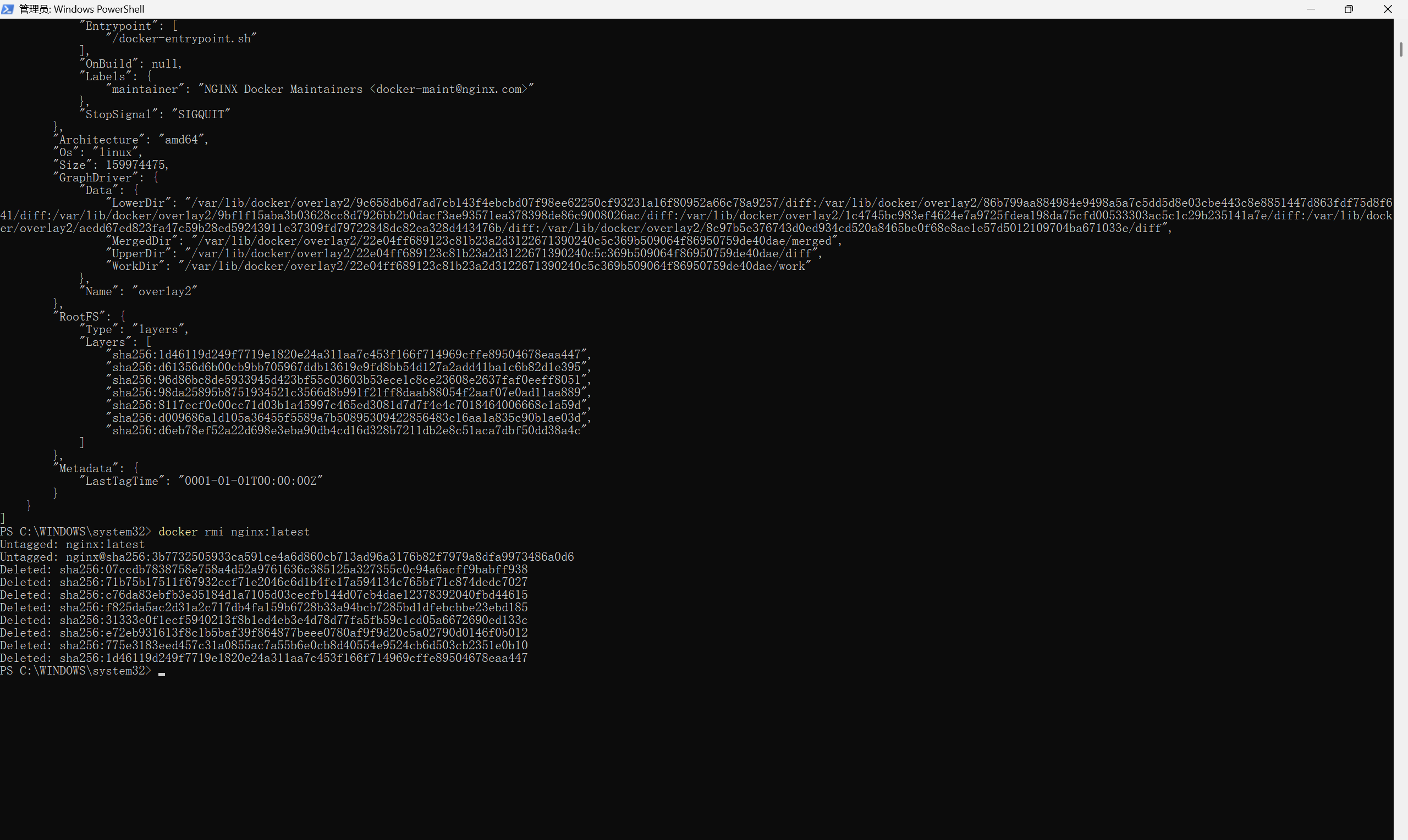
docker images



docker inspect nginx:latest



docker rmi nginx:latest



**Image layering example:**

nginx:latest

├── Layer 4: Configuration File

├── Layer 3: Nginx Binary files

├── Layer 2: System Tools

└── Layer 1: Basic operating system

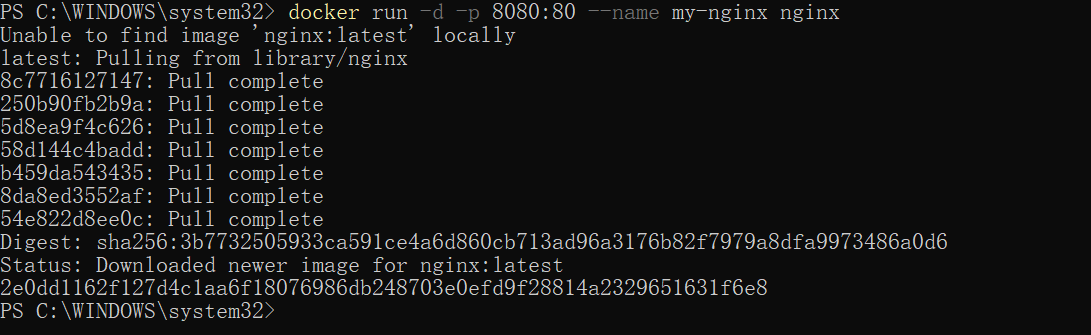
**3：Publishing Ports**

**Concepts：**

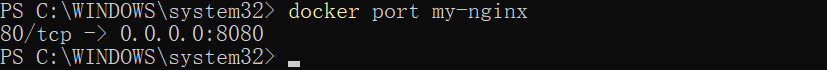
* + Mapping internal ports of the container to host ports
  + Making services within the container accessible from outside

**Experimental procedures：**

docker run -d -p 8080:80 --name my-nginx nginx



docker port my-nginx



**Port mapping syntax: -p host port:container port**

-p 80:80

-p 8080:80

-p 192.168.100.1:8080:80

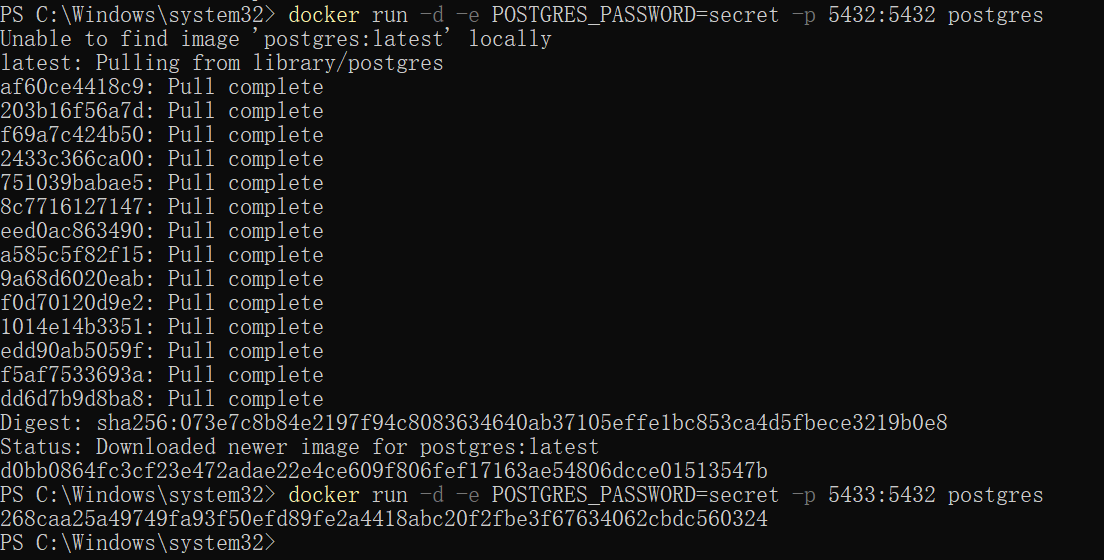
**4：Overriding container defaults**

**Concepts：**

* **Override default commands and environment variables defined in the image**
* **Customize container runtime behavior**

**Experimental procedures：**

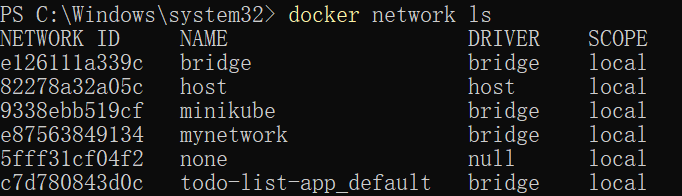
1. Start a container using the Postgres image with the following command:
2. Start a second Postgres container mapped to a different port.



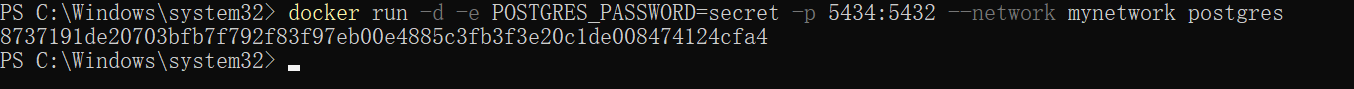
1. Run Postgres container in a controlled network

Create a new custom network by using the following command:

Verify the network by running the following command:

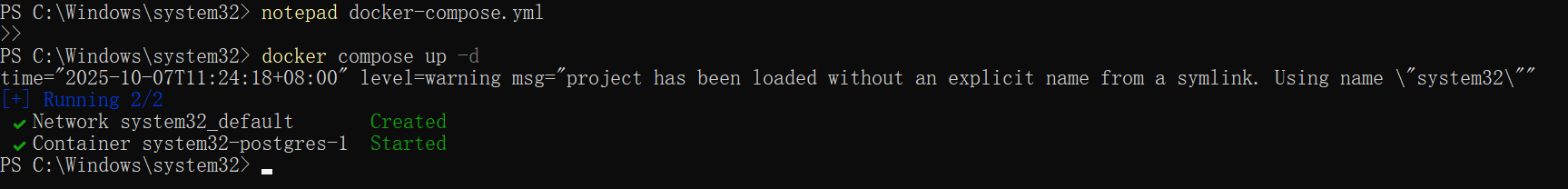


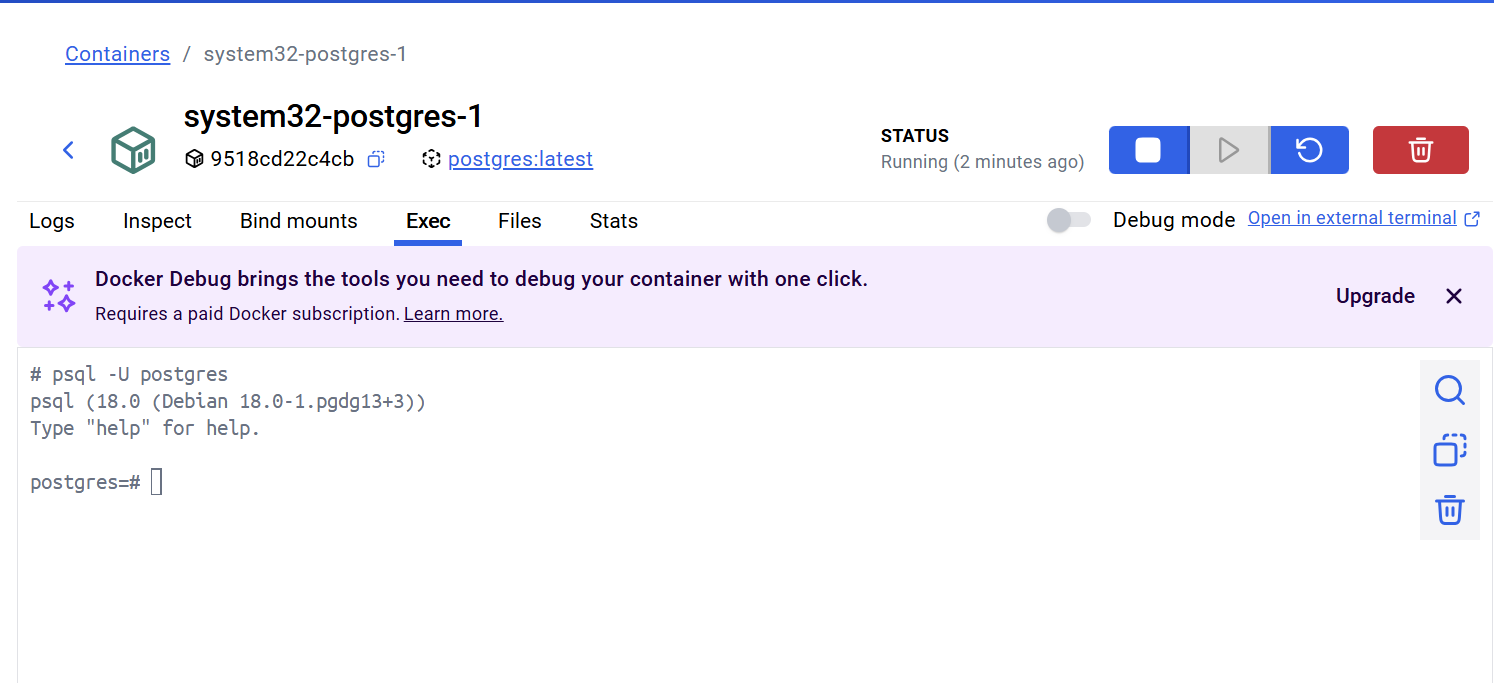
Connect Postgres to the custom network by using the following command:



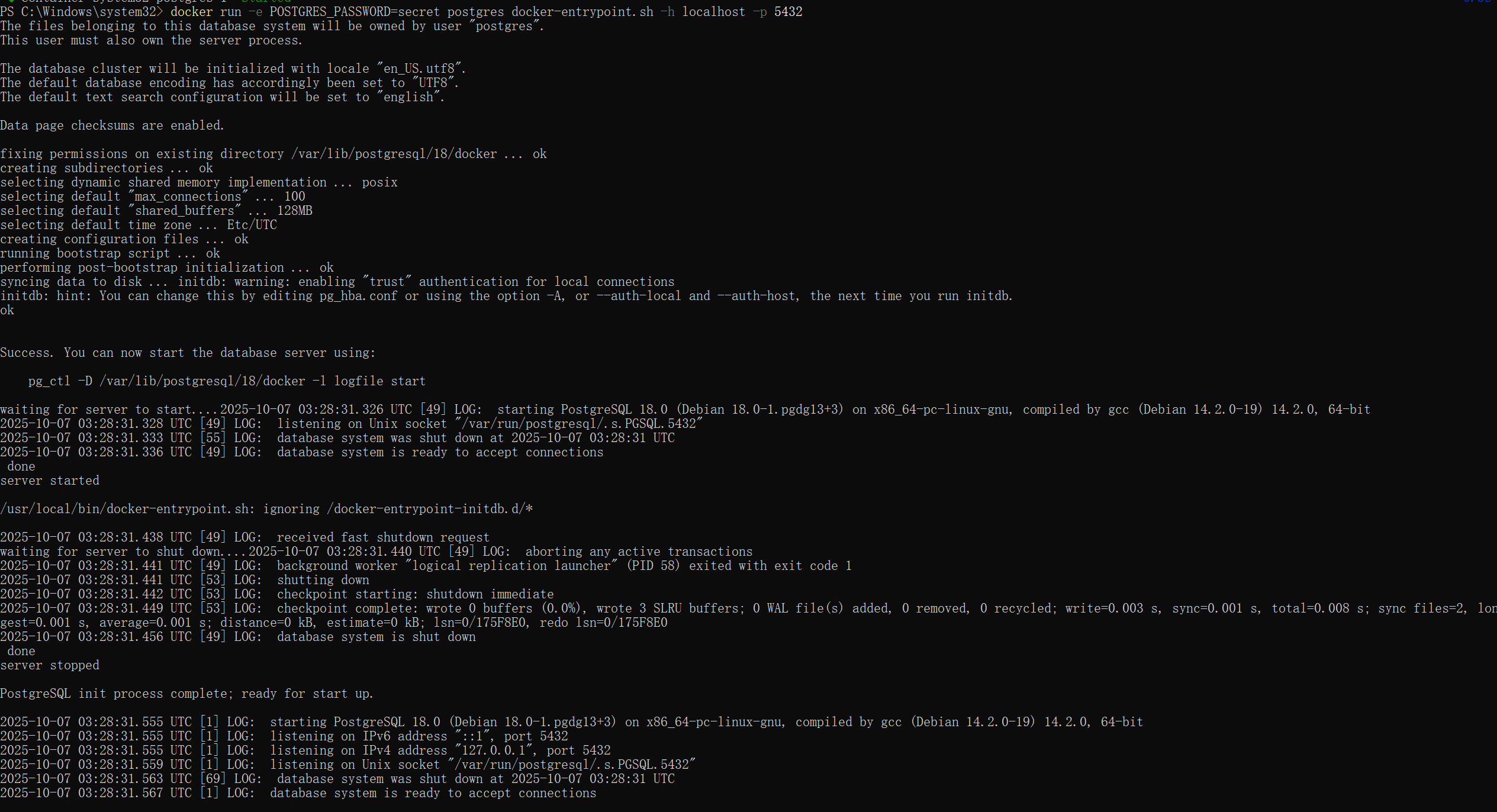
1. Manage the resources

This is where the docker run command shines again. It offers flags like --memory and --cpus to restrict how much CPU and memory a container can use.





1. Override the default CMD and ENTRYPOINT with docker run



**5：Persisting Data**

**Concept：**

* + Use volumes and bind mounts to persist data
  + Avoid data loss when containers are deleted

**Experimental procedures：**

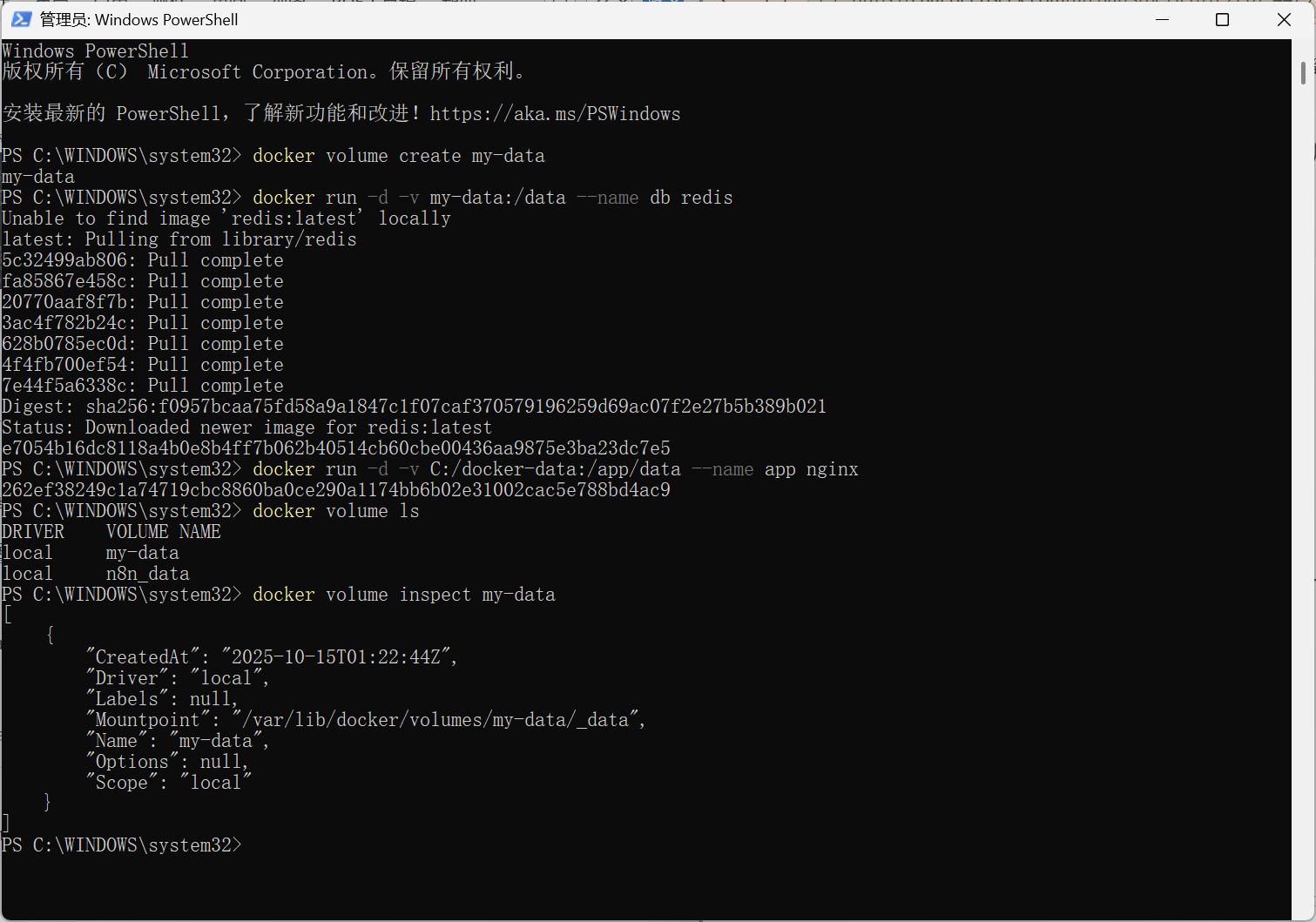
docker volume create my-data

docker run -d -v my-data:/data --name db redis

docker run -d -v C:/docker-data:/app/data --name app nginx

docker volume ls

docker volume inspect my-data



**6：Sharing local files**

**Concept：**

* + Mount host directories into containers
  + Enable real-time code synchronization during development

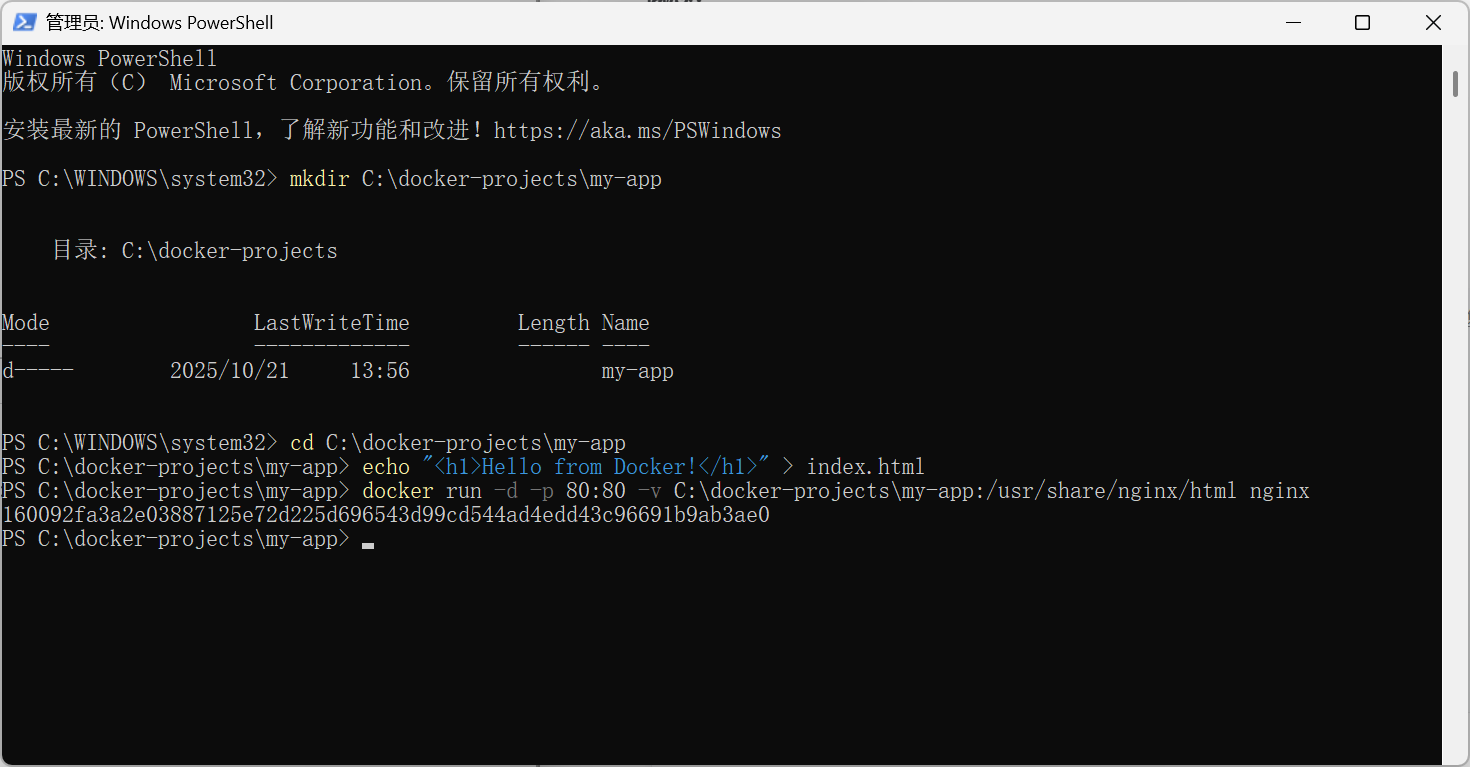
**Experimental procedures:**

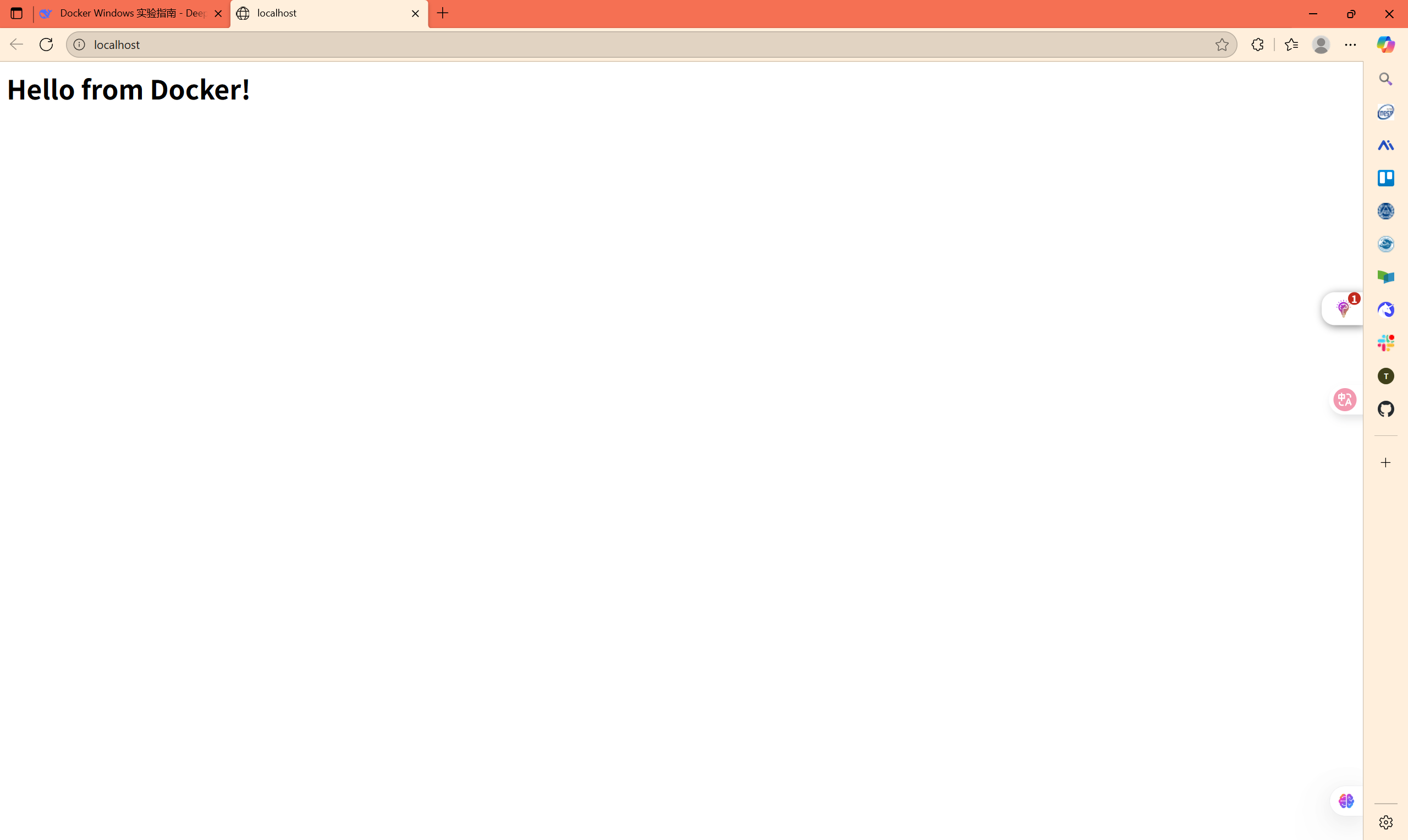
mkdir C:\docker-projects\my-app

cd C:\docker-projects\my-app

echo "<h1>Hello from Docker!</h1>" > index.html

docker run -d -p 80:80 -v C:\docker-projects\my-app:/usr/share/nginx/html nginx





**7：Multi-container applications (Docker Compose)**

**Concept：**

* + Define and run multi-container applications using docker-compose.yml
  + Simplify deployment of complex applications

**Project Structure:**

1. **docker-compose.yml：**

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "8080:80"

volumes:

- ./html:/usr/share/nginx/html

depends\_on:

- db

db:

image: redis:alpine

volumes:

- redis-data:/data

command: redis-server --appendonly yes

volumes:

redis-data:

1. **Start the application:**

mkdir docker-compose-demo

cd docker-compose-demo

mkdir html

echo "<h1>Multi-Container App</h1>" > html/index.html

docker-compose up -d

docker-compose ps

docker-compose down