**Introduction to Docker**

Docker is a platform for developing, shipping, and running applications in containers. Containers allow you to package an application with all of its dependencies into a standardized unit for software development.

Key Benefits:

Consistency: Same environment across development, testing, and production

Isolation: Applications run in isolated environments

Portability: Run anywhere Docker is installed

Efficiency: Lightweight compared to virtual machines

<https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/>

What is a Container?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Container Characteristics:

Isolated process：running on a host machine

Shares the host OS kernel

Contains application code, runtime, system tools, system libraries, and settings\*\*

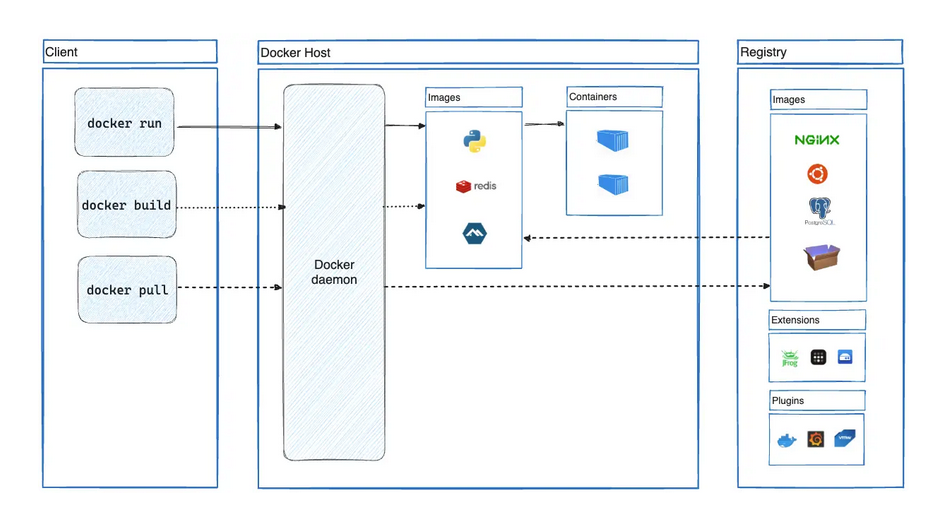
Example: Running a Simple Container

Run an nginx container：docker run -d --name web-server nginx:latest

Check running containers：docker ps

View container logs：docker logs web-server

[Container Architecture]



Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

Docker objects

<https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/>

Image

A Docker image is a read-only template with instructions for creating a Docker container. It includes everything needed to run an application - the code, runtime, libraries, environment variables, and configuration files.

Image Layers:

Base Image: The foundation layer (e.g., Ubuntu, Alpine Linux)

Additional Layers: Each instruction in a Dockerfile creates a new layer

Read-only: Images are immutable once built

<https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-docker-compose/>

What is Docker Compose?

Docker Compose is a tool for defining and running multi-container Docker applications. You use a YAML file to configure your application's services, networks, and volumes.

Key Features:

Single command to start multiple services

Service dependencies and startup order

Volume and network management

Environment variable configuration

<https://docs.docker.com/get-started/docker-concepts/running-containers/publishing-ports/>

Publishing ports allows you to access container services from the host machine or external network.

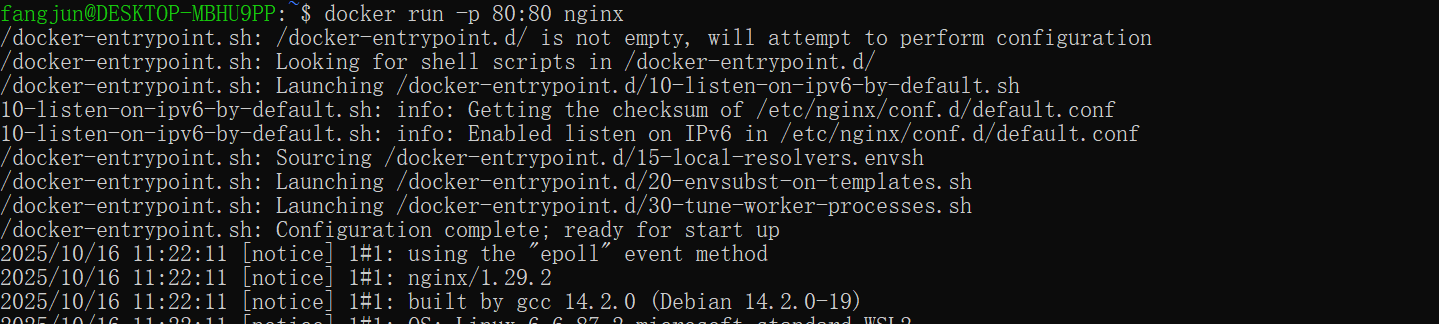
Port Publishing Methods:

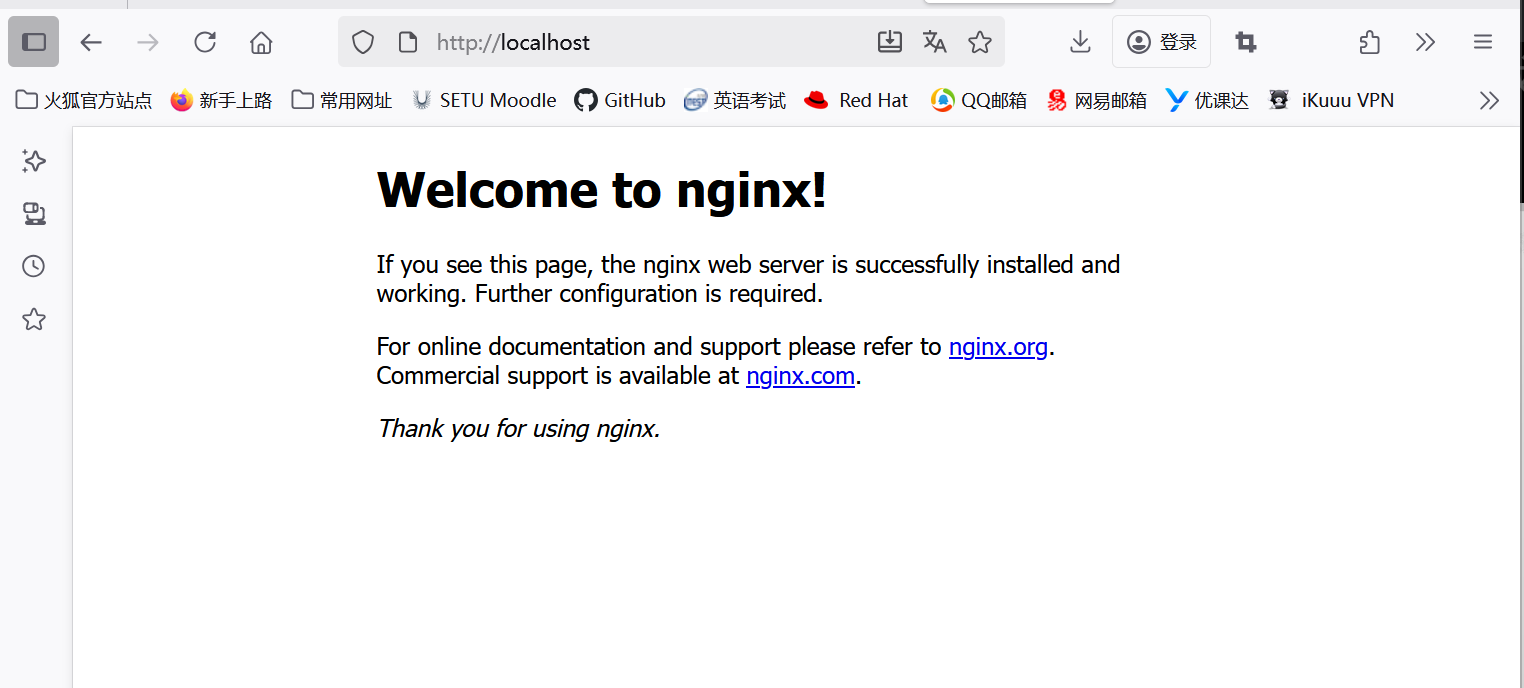
Map specific ports：docker run -p 8080:80 nginx

Map to any available host port：docker run -p 80 nginx

Map multiple ports：docker run -p 8080:80 -p 3000:3000 my-app

Map UDP ports：docker run -p 53:53/udp dns-server





<https://docs.docker.com/get-started/docker-concepts/running-containers/overriding-container-defaults/>

Sometimes you might want to use separate database instances for development and testing purposes. Running these database instances on the same port might conflict. You can use the -p option in docker run to map container ports to host ports, allowing you to run the multiple instances of the container without any conflict.

You can override default container settings using command-line options and environment variables.

docker run -d -p HOST\_PORT:CONTAINER\_PORT postgres

<https://docs.docker.com/get-started/docker-concepts/running-containers/persisting-container-data/>

Volumes are a storage mechanism that provide the ability to persist data beyond the lifecycle of an individual container. Think of it like providing a shortcut or symlink from inside the container to outside the container.

Docker provides several ways to persist data beyond the container's lifecycle.

Data Persistence Methods:

1. Volumes(Managed by Docker)

2. Bind Mount (Host file system paths)

3. tmpfs Mounts(In-memory storage)

<https://docs.docker.com/get-started/docker-concepts/running-containers/sharing-local-files/>

Both -v (or --volume) and --mount flags used with the docker run command let you share files or directories between your local machine (host) and a Docker container. However, there are some key differences in their behavior and usage.

The -v flag is simpler and more convenient for basic volume or bind mount operations. If the host location doesn’t exist when using -v or --volume, a directory will be automatically created.

Imagine you're a developer working on a project. You have a source directory on your development machine where your code resides. When you compile or build your code, the generated artifacts (compiled code, executables, images, etc.) are saved in a separate subdirectory within your source directory. In the following examples, this subdirectory is /HOST/PATH. Now you want these build artifacts to be accessible within a Docker container running your application. Additionally, you want the container to automatically access the latest build artifacts whenever you rebuild your code.

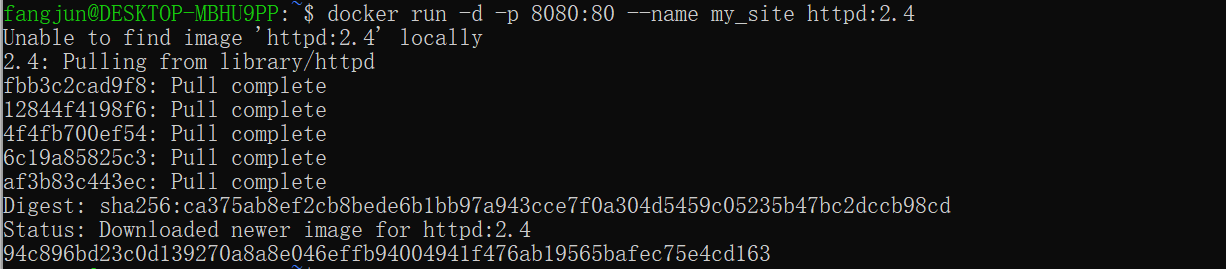
docker run --mount type=bind,source=/HOST/PATH,target=/CONTAINER/PATH,readonly nginx

Bind mounts allow you to share files between the host and container, which is especially useful for development.

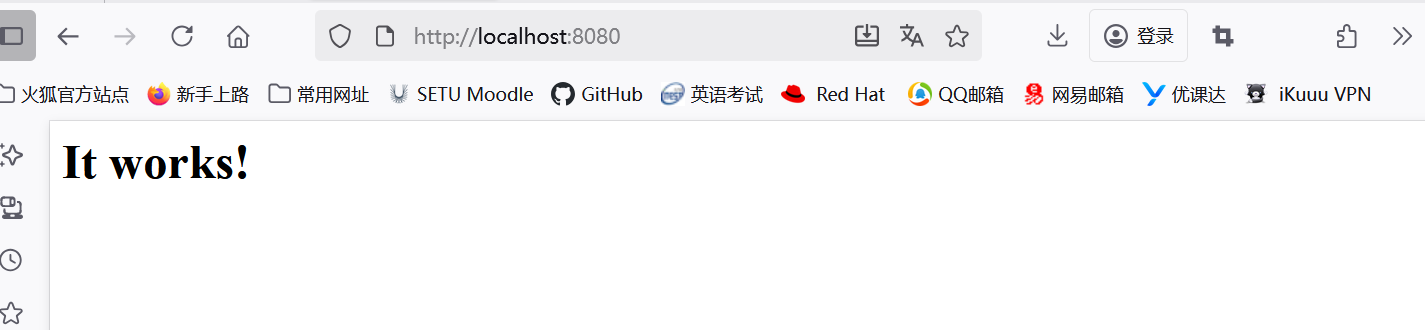
When using bind mounts, it's crucial to ensure that Docker has the necessary permissions to access the host directory. To grant read/write access, you can use the :ro flag (read-only) or :rw (read-write) with the -v or --mount flag during container creation. For example, the following command grants read-write access permission.

docker run -v HOST-DIRECTORY:/CONTAINER-DIRECTORY:rw nginx

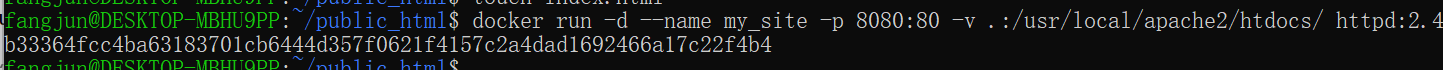
[Run a container](https://docs.docker.com/get-started/docker-concepts/running-containers/sharing-local-files/" \l "run-a-container)

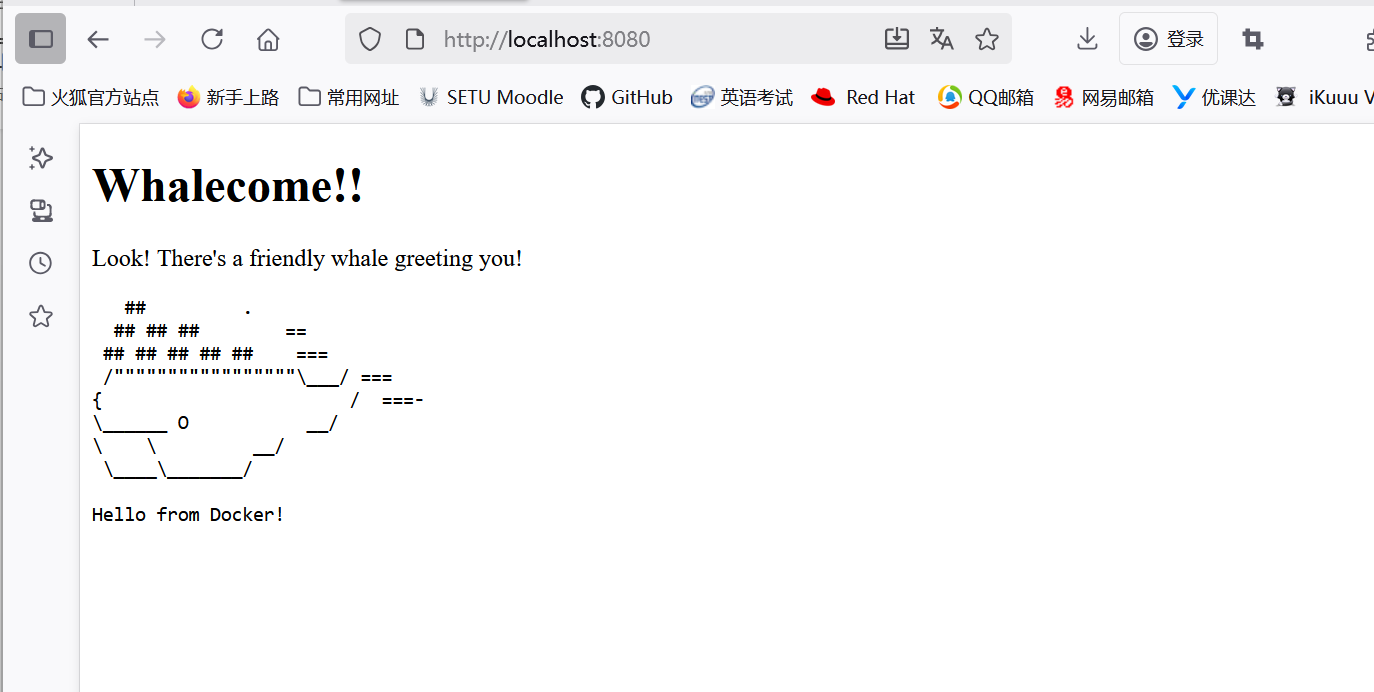


Open the browser and access http://localhost:8080：

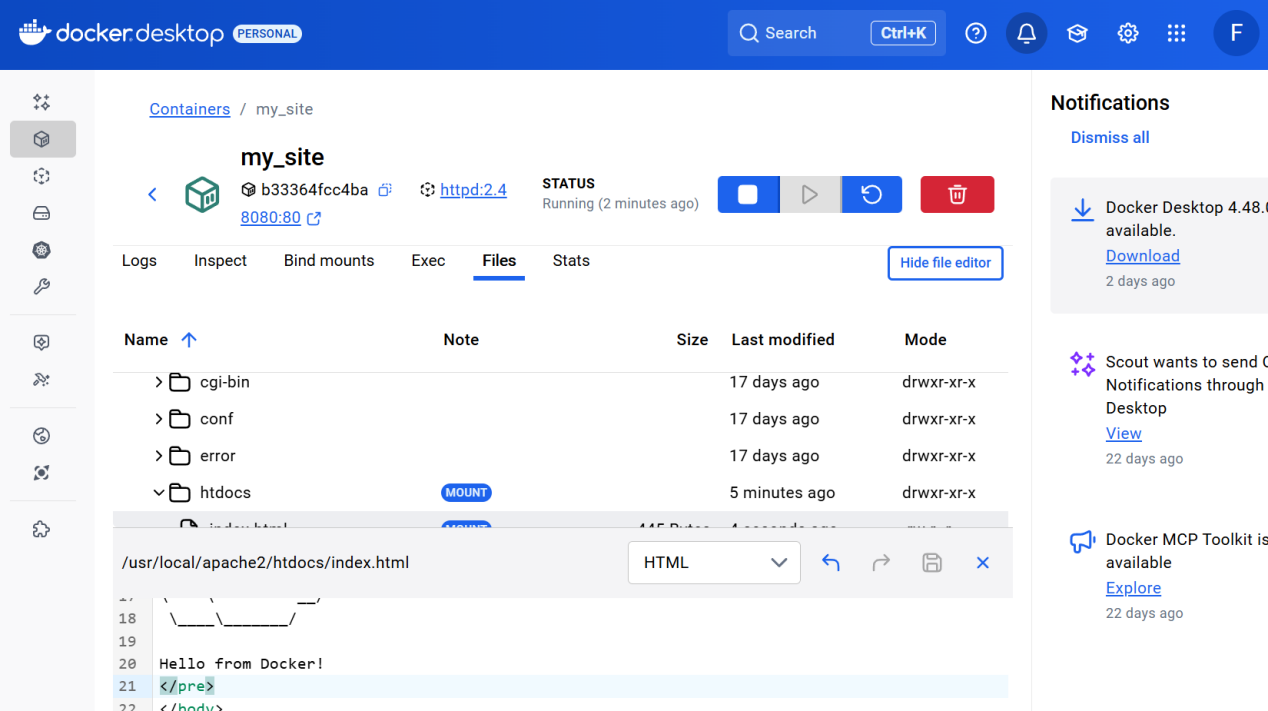


Use a bind mount：Navigate into the newly created directory public\_html and create a file called index.html with the following content. This is a basic HTML document that creates a simple webpage that welcomes you with a friendly whale.





Access the file on the Docker Desktop Dashboard



<https://docs.docker.com/get-started/docker-concepts/running-containers/multi-container-applications/>

Multi-Container Applications

Complex applications often require multiple services working together. Docker Compose makes it easy to manage these relationships.

Conclusion

Docker provides a powerful platform for containerization that simplifies application deployment and development. By understanding containers, images, Docker Compose, and various runtime configurations, you can create robust and portable applications.

Recommendations:

https://docs.docker.com/desktop/install/windows-install/

https://docs.docker.com/desktop/wsl/