Docker

Containerization & Application Deployment

Complete Guide to Modern Container Technology

What is Docker?

Definition

Docker is an open-source platform that automates the deployment, scaling, and management of applications by using containerization. It packages an application and its dependencies into a standardized unit called a container.

Real-World Analogy

Think of it like a shipping container. Just as a shipping container can hold any type of cargo and be transported by ship, train, or truck, a Docker container can hold any application and run on any machine that has Docker installed.

Docker Core Concepts

Three Key Pillars

Package

Bundles the application code, runtime, system tools, and libraries into one standardized unit.

Isolate

Containers are isolated from each other and from the host machine's operating system.

Lightweight

They share the host machine's OS kernel, making them much more efficient than traditional Virtual Machines (VMs).

Why Docker?

Key Benefits

Consistency & Portability

A Docker container runs the same way regardless of where it is deployed. This eliminates inconsistencies between development, testing, and production environments.

Efficiency & Performance

Containers are incredibly lightweight. They start faster and use significantly fewer resources (CPU, RAM, and storage) compared to Virtual Machines.

Isolation & Security

Containers provide process-level isolation. An application in one container cannot interfere with another, enhancing security and stability.

Rapid Deployment & Scalability

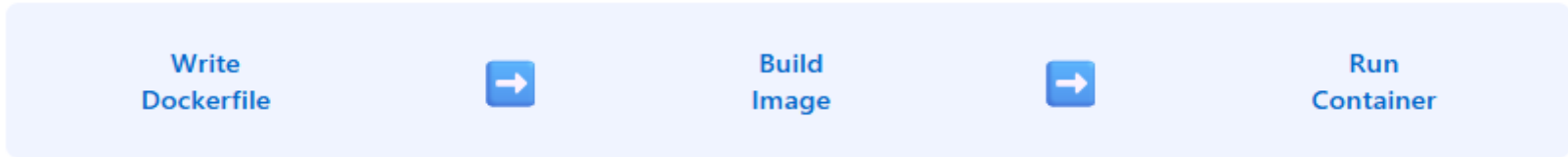
Build a container image once and deploy it anywhere in seconds. Scaling is as simple as running more containers.

Microservices Architecture

Each service can be developed, deployed, and scaled independently in its own container, simplifying complex applications.

Docker Workflow

Write • Build • Run



Write a Dockerfile

A text file containing instructions on how to build a Docker image. It specifies the base OS, application code, commands, and configuration details.

Build a Docker Image

Use the **docker build** command to create a Docker image. An image is a read-only template containing the application and its environment.

Run a Docker Container

Use the **docker run** command to launch a container from an image. A container is the runnable instance of your application.

Docker Architecture

Docker uses a client-server architecture where the Docker client communicates with the Docker daemon (server).

Docker Daemon (dockerd)

The Docker engine, a persistent background process that manages Docker objects like images, containers, networks, and volumes. The daemon listens for API requests from the Docker client.

Docker Client (docker)

The primary user interface to Docker. It's a command-line tool used to interact with the Docker daemon. When you type commands like **docker run** or **docker build**, the client sends them to dockerd.

Docker Registry

A storage system for Docker images. Docker Hub is the default and largest public registry, but you can also run your own private registry. When you run **docker pull**, images are fetched from the registry.

Docker Objects

Images

Read-only templates used to create containers. Immutable blueprints of your application.

Containers

Runnable instances of images. The live, executing version of your application.

Volumes

The preferred mechanism for persisting data generated by and used by Docker containers.

Networks

Provide a way to isolate containers and allow them to communicate with each other securely.

How Docker Works

Step-by-Step Execution

Step 1

Developer uses the Docker Client to send a command (e.g., **docker run my-app**).

Step 2

The client sends this request to the Docker Daemon.

Step 3

The daemon checks if it has the my-app image locally. If not, it pulls the image from the Docker Registry.

Step 4

The daemon creates and runs a new container based on that image.

Hands-on: Setup & Prerequisites

Getting Started

Prerequisites

→ Install Docker Desktop on your machine from the official Docker website

→ Ensure you have Python 3.8+ installed for our example application

→ Basic knowledge of command line or terminal

→ Text editor for creating files

What We'll Build

A simple "Hello World" API using FastAPI framework, then containerize it with Docker.

→ Create a Python FastAPI web server

→ Write a Dockerfile

→ Build a Docker image

→ Run and test the container

Step 1: Create FastAPI Application

Create two files in your project folder:

requirements.txt:

```
fastapi==0.104.1 uvicorn==0.24.0
```

main.py:

```
from fastapi import FastAPI app = FastAPI() @app.get("/") def read_root(): return {"message": "Hello, Docker World! 🐳"} @app.get("/api/hello/{name}") def read_hello(name: str): return {"message": f"Hello, {name}!"} if __name__ == "__main__": import uvicorn uvicorn.run(app, host="0.0.0.0", port=8080)
```

Step 2: Write a Dockerfile

Create a file named Dockerfile (no extension):

```
# Specify the base image FROM python:3.11-slim # Set the working directory inside the container
WORKDIR /app # Copy requirements file COPY requirements.txt . # Install dependencies RUN pip install --
-no-cache-dir -r requirements.txt # Copy application source code COPY . . # Expose the port EXPOSE
8080 # Define the startup command CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Each line in the Dockerfile is a layer that Docker builds. This structure optimizes build time through caching.

Step 3: Build the Docker Image

Open terminal and navigate to your project folder:

```
docker build -t hello-docker-app .
```

Command Breakdown

→ **docker build** - Instructs Docker to build an image

→ **-t hello-docker-app** - Tags or names your image

→ **.** - Tells Docker to look for Dockerfile in current directory

Docker will execute each instruction in the Dockerfile and create an image layer by layer.

Step 4: Run the Docker Container

Once the image is built, run it as a container:

```
docker run -p 4000:8080 -d hello-docker-app
```

Flag Explanations

→ **-p 4000:8080** - Maps port 4000 on your host to port 8080 in container

→ **-d** - Runs the container in detached mode (background)

→ **hello-docker-app** - Name of the image to run

Step 5: Verify Your Application

Your Docker container should now be running!

Test the Application

Open your web browser or API client and navigate to:

```
http://localhost:4000
```

You should see: **{"message":"Hello, Docker World! 🐳 "}**

Or access the interactive API docs at:

```
http://localhost:4000/docs
```

Try the endpoint: **GET /api/hello/YourName**

Congratulations!

You have successfully created, built, and deployed your first Dockerized application!