# Understanding Containerisation with Docker: Deploying a Python “Hello World” Application

## Introduction: What is Containerisation?

Software development often faces the challenge: *"it works on my machine, but not yours."* This usually happens because of differences in operating systems, libraries, or other dependencies. ***Containerisation*** solves this by packaging an application along with its entire runtime environment – all necessary code, libraries, and configuration files – into a single, ***standardised*** unit called a container. This ensures the application runs consistently and reliably, regardless of where the container is deployed.

What is Docker?

Docker is currently the most popular platform for creating, deploying, and managing containers. It provides tools to build container images (blueprints for containers) and run them efficiently. Compared to traditional virtual machines, Docker containers are more lightweight, start faster, and use fewer system resources because they share the host operating system's kernel.

Why Use Docker?

* **Consistency**: Ensures applications run identically across development, testing, and production environments.
* **Portability**: Containers can be easily moved and run on any machine with Docker installed.
* **Isolation**: Applications inside containers run independently, preventing conflicts.
* **Scalability**: Easily scale applications by running multiple instances of a container.

Docker "Hello World" Example with Python

Let's create a simple Docker container for a Python *"Hello World"* application.

### Step 1: Create the Python Script

Create a file named *app.py*:

# app.py

print("Hello, Docker World!")

### Step 2: Create the Dockerfile

A Dockerfile is a text file containing instructions to build a Docker image. Create a file named *Dockerfile* (this is the standard convention):

# Dockerfile

# Use an official lightweight Python image

FROM python:3.12-slim

# Set the working directory inside the container

WORKDIR /app

# Copy the Python script into the container's working directory

COPY app.py .

# Specify the command to run when the container starts

CMD ["python", "app.py"]

### Step 3: Build the Docker Image

In your terminal, navigate to the directory containing *app.py* and *Dockerfile*, then run:

# Build the image and tag it as 'hello-docker'

docker build -t hello-docker .

### Step 4: Run the Docker Container

Execute the following command to run your newly built image as a container:

docker run hello-docker

You should see the output:

Hello, Docker World!

## Beyond "Hello World": Scaling Up

While this example is basic, Docker's power shines in more complex scenarios. For applications with multiple components, *docker-compose* allows you to define and manage multi-container applications easily. This simplifies handling different sets of dependencies for various projects running on the same system, making development and deployment significantly more manageable.

## Learn More

If you would like to learn more about applying Docker to your project, Research IT is hosting the [Introduction to Docker](http://app.manchester.ac.uk/RDOCKER) course on 6-7 October 2025. The course notes, detailing the content, can be found [here](https://uomresearchit.github.io/docker-introduction/). You can find details on this and other courses offered by Research IT [here](https://www.staffnet.manchester.ac.uk/talent-development/learning-pathways/professional-and-technical-development/digital-skills/research-computing/).