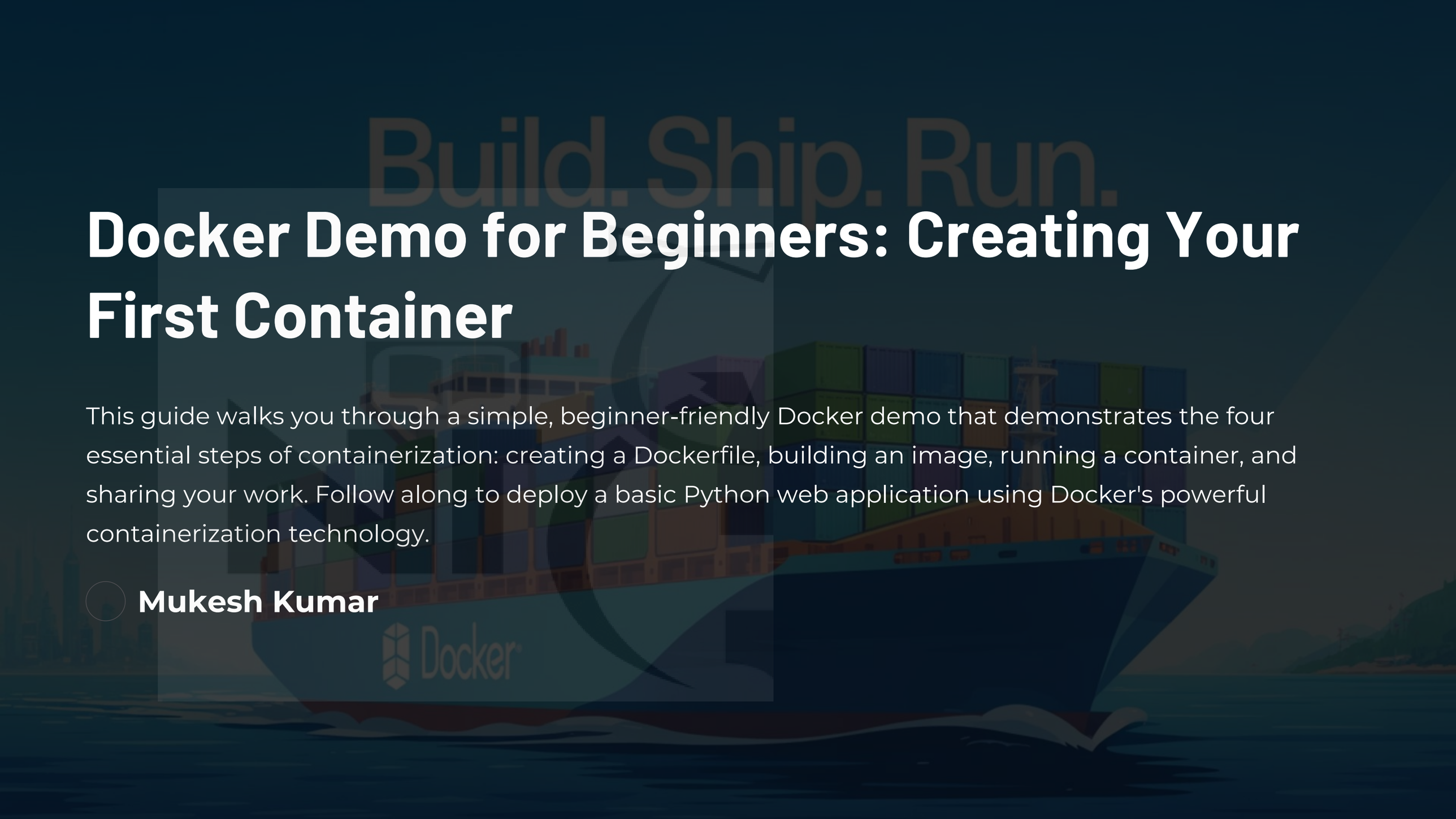


Docker Demo for Beginners: Creating Your First Container

This guide walks you through a simple, beginner-friendly Docker demo that demonstrates the four essential steps of containerization: creating a Dockerfile, building an image, running a container, and sharing your work. Follow along to deploy a basic Python web application using Docker's powerful containerization technology.

Mukesh Kumar

Introduction and Goal

Simple Demo

A beginner-friendly introduction to Docker containerization

Python Web App

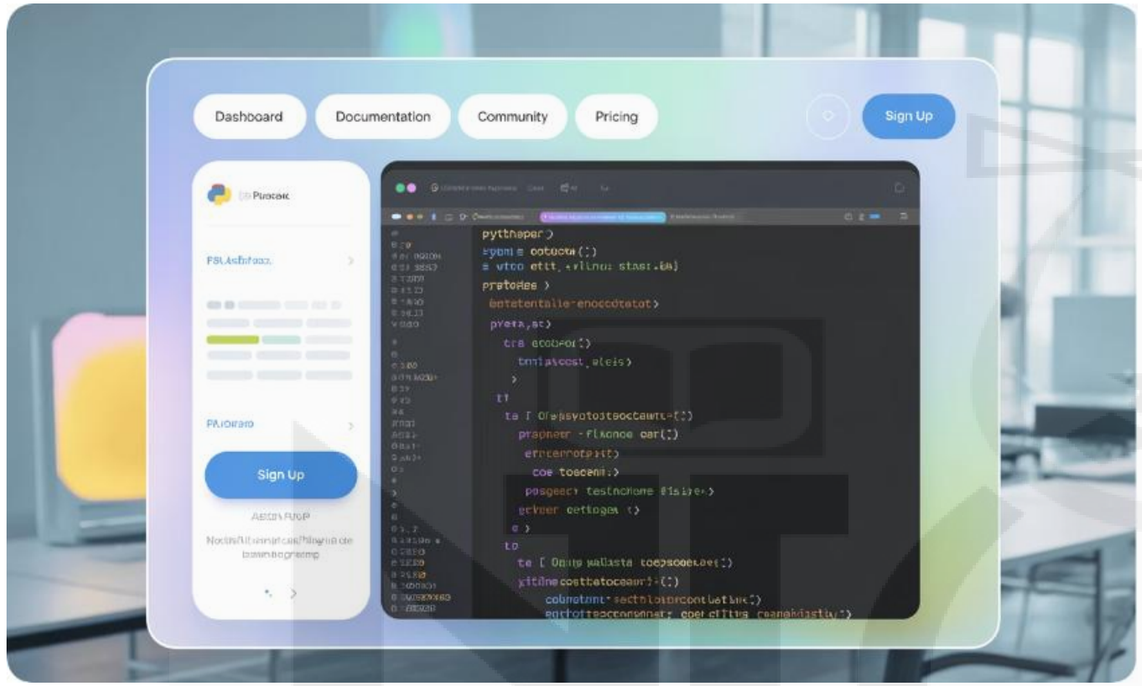
Build a minimal Flask application that serves a greeting

Docker Basics

Learn the fundamental Docker workflow from Dockerfile to sharing

This demo aims to introduce you to Docker concepts in the most straightforward way possible. By containerizing a simple Python web application, you'll experience firsthand how Docker enables consistent, portable application deployment. The process illustrates the core steps of Docker development while requiring minimal setup and coding knowledge.

Step 1: Create the Application



Creating a Basic Flask Web Application

Our first step is to create a simple Python web application using the Flask framework. This minimal application will serve a "Hello from Docker!" message when accessed through a web browser.

Create a file named **app.py** with the following code:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from Docker!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

This code creates a basic web server that listens on all interfaces (0.0.0.0) on port 5000, making it accessible when containerized.

Step 2: Create a Dockerfile

Writing the Dockerfile

The Dockerfile is a text document containing instructions for building your Docker image. It specifies everything needed to run your application, from the base operating system to application code and dependencies.

Create a file named **Dockerfile** (with no file extension) in the same directory as your app.py file:

```
# Use Python base image
FROM python:3.9

# Set working directory
WORKDIR /app

# Copy files to the container
COPY . .

# Install dependencies
RUN pip install flask

# Run the app
CMD ["python", "app.py"]
```

Each instruction in the Dockerfile creates a layer in the image:

- **FROM**: Specifies the base image (Python 3.9)
- **WORKDIR**: Sets the working directory inside the container
- **COPY**: Transfers your files into the container
- **RUN**: Installs the Flask dependency
- **CMD**: Defines the command to run when the container starts

Step 3: Build the Docker Image

Open Terminal

Navigate to your project directory

Run Build Command

Execute: docker build -t my-docker-demo .

Process Layers

Docker processes each instruction in your Dockerfile

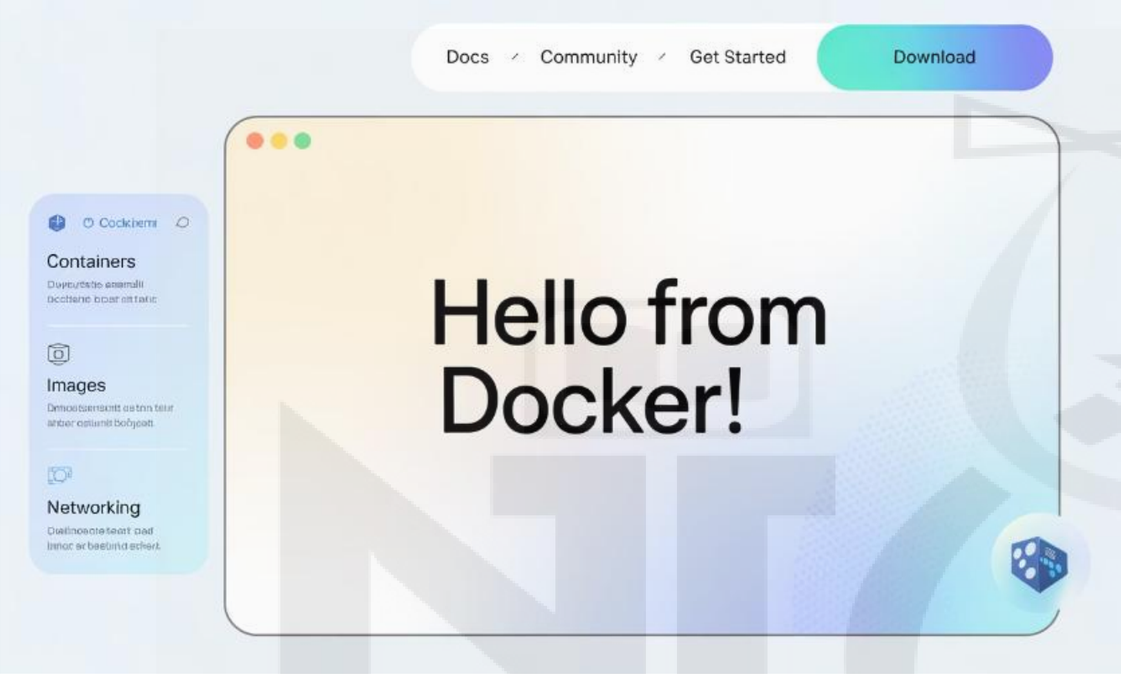
Image Created

Your image is now ready to run

The **docker build** command transforms your Dockerfile and application code into a Docker image. The **-t** flag tags your image with a name (in this case, "my-docker-demo"), making it easy to reference later. The period (.) at the end tells Docker to look for the Dockerfile in the current directory.

During the build process, Docker will show the progress of each step defined in your Dockerfile. It downloads the base Python image, sets up your working environment, copies your files, installs Flask, and prepares your application to run.

Step 4: Run the Container



After running the container, open your web browser and navigate to **http://localhost:5000**. You should see your application displaying the message "Hello from Docker!"

Running Your Docker Container

Now that your image is built, you can run it as a container with this command:

```
docker run -p 5000:5000 my-docker-demo
```

The **docker run** command creates and starts a container from your image. The **-p 5000:5000** flag maps port 5000 from your container to port 5000 on your host machine, making the web application accessible.

This port mapping is crucial—without it, your application would run inside the container but wouldn't be accessible from your browser. The format is **-p host_port:container_port**.

Your terminal will show logs from the Flask application running inside the container. To stop the container, press **Ctrl+C** in your terminal window.

Step 5: Share Your Docker Image (Optional)

Create a Docker Hub Account

Sign up at hub.docker.com if you don't already have an account. This is Docker's official repository for sharing and distributing container images.

Tag Your Image

Tag your local image with your Docker Hub username:

```
docker tag my-docker-demo your_dockerhub_username/my-docker-demo
```

Push to Docker Hub

Upload your tagged image to Docker Hub:

```
docker push your_dockerhub_username/my-docker-demo
```

After pushing your image to Docker Hub, anyone can download and run your containerized application with a single command:

```
docker run -p 5000:5000 your_dockerhub_username/my-docker-demo
```

This powerful feature demonstrates the true value of Docker - the ability to package your application with all its dependencies into a single unit that can run consistently anywhere Docker is installed.