Docker & Containerised Workflows

Master containerisation for consistent, portable, and scalable application deployment across any environment

Why Docker Matters

Consistency Everywhere

Deploy the same environment across development, staging, and production—eliminating the classic "it works on my machine" problem once and for all.

Perfect Isolation

Run multiple applications or services side by side without dependency conflicts, version clashes, or resource competition between projects.

True Portability

Containers run identically on any machine with Docker installed—from your laptop to cloud servers, maintaining complete environment parity.

Before diving in, understand how Docker integrates with CI/CD pipelines, DevOps workflows, and orchestration platforms. Recognise when containerisation adds value versus simpler virtual environments.

Docker Fundamentals

Core Concepts

Images vs Containers: Images are blueprints; containers are running instances of those images.

Dockerfile Basics: Step-by-step instructions that define how to build your image, from base OS to application setup.

Essential Commands: Master `docker run`, `docker ps`, and `docker stop` to control container lifecycles effectively.

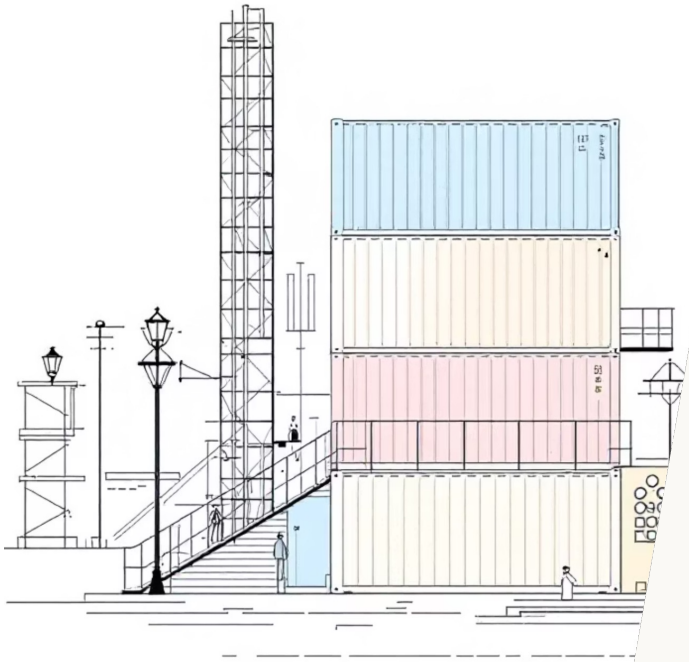
Data & Networking

Volumes: Persist data beyond container lifecycles, ensuring your databases and files survive restarts and updates.

Container Networking: Enable secure communication between containers whilst maintaining isolation from the host system.

Learning Resources

Docker Tutorial for Beginners • Complete Docker Course • Docker in 100 Seconds



Docker Compose: Multi-Container Applications

01

Define Services

Specify all containers, networks, and volumes in a single `docker-compose.yml` file for your entire application stack.

02

Configure Relationships

Link containers together, manage environment variables, and define dependencies between services with declarative syntax.

03

Scale & Deploy

Launch your entire application with one command, scale services horizontally, and orchestrate complex multi-tier architectures.

Docker Compose transforms complex multi-container setups into reproducible, version-controlled configurations that your entire team can share and deploy consistently.

Video Tutorials

Docker Compose Tutorial • Docker Compose Deep Dive • Practical Docker Compose

CI/CD Pipeline Integration

Automated Builds

Use GitHub Actions to automatically build Docker images when code is pushed, ensuring every commit is containerised.

Testing Stage

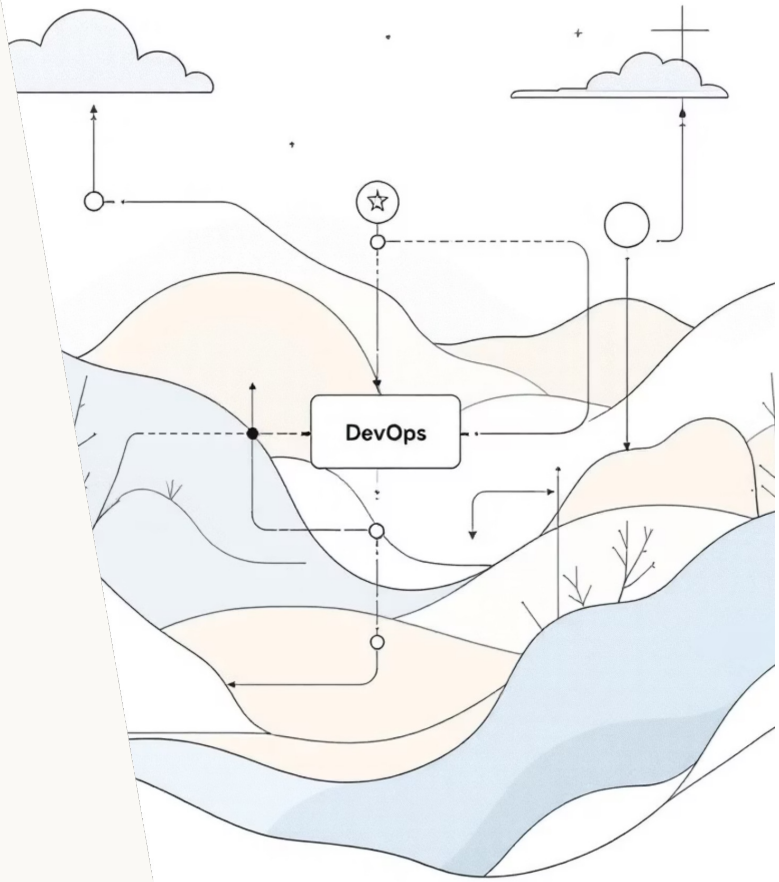
Run automated tests inside containers before deployment, catching issues in an environment identical to production.

Deploy Anywhere

Push verified images to registries and deploy seamlessly to cloud platforms or Kubernetes clusters.

Watch & Learn

GitHub Actions Docker Tutorial • Docker CI/CD Pipeline • Automated Docker Workflows



Architectural Thinking for Docker

Map Your Architecture

Before writing any Dockerfile, sketch out which services need isolation, how they communicate, and what resources they share.

Plan Data Strategy

Identify which data needs persistence, which containers share volumes, and how to handle backups and migrations effectively.

Design for Scale

Consider how services will scale horizontally, how failures are handled, and how rolling updates maintain zero downtime.

Network Architecture

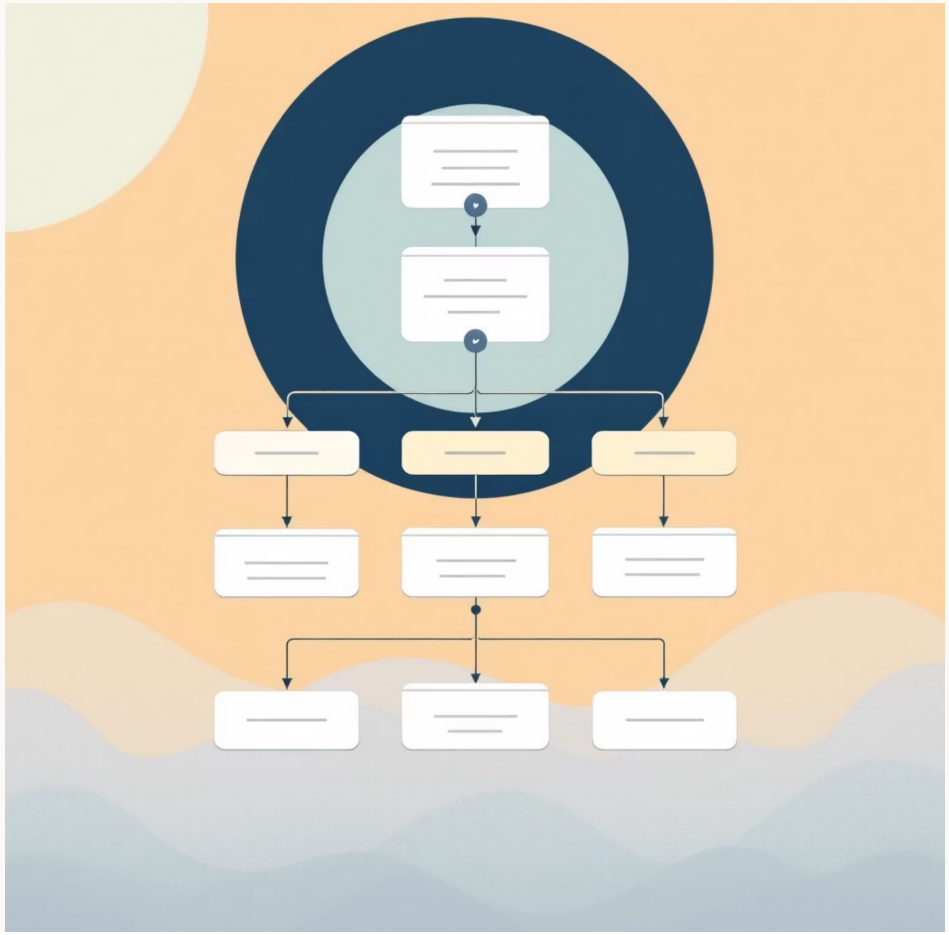
Define container networking requirements, security boundaries, and how services discover and communicate with each other.

Thoughtful planning before implementation saves countless hours debugging container configurations and prevents architectural debt.

Real-World Docker Patterns

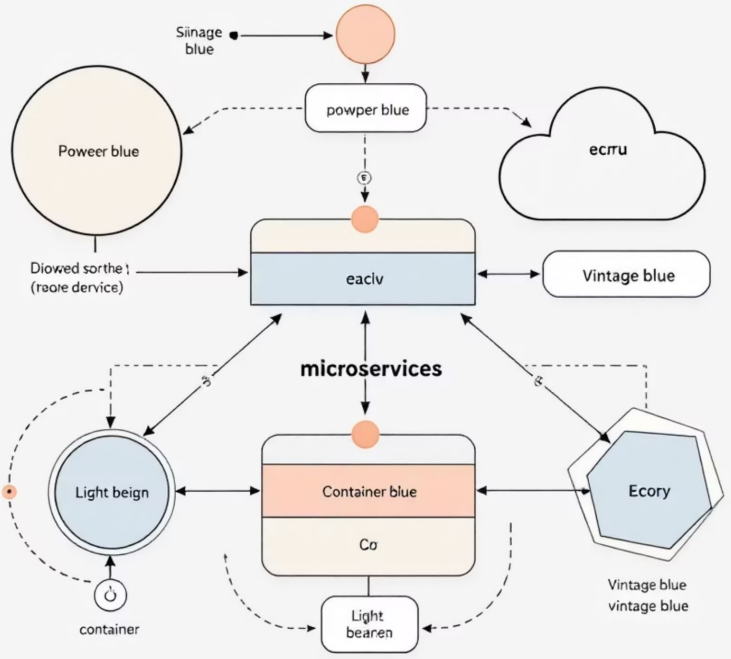
Web Application Stack

Flask or Django frontend + PostgreSQL database, connected via Docker network with persistent volume for data storage.



Microservices Architecture

API gateway, worker services, and message queue (RabbitMQ/Redis) orchestrated with Docker Compose for scalable systems.



Common Docker Workflows

1 Build Image

Create Dockerfile, define base image, install dependencies, copy application code

2 Test Locally

Run container, verify functionality, check logs, iterate on configuration

3 Push to Registry

Tag image appropriately, authenticate with registry, push for distribution

4 Deploy

Pull image on target server, run with production configs, monitor health

Pro tip: Use multi-stage builds to keep production images lean by separating build dependencies from runtime requirements.

Best Practices & Pitfalls

Keep Images Small

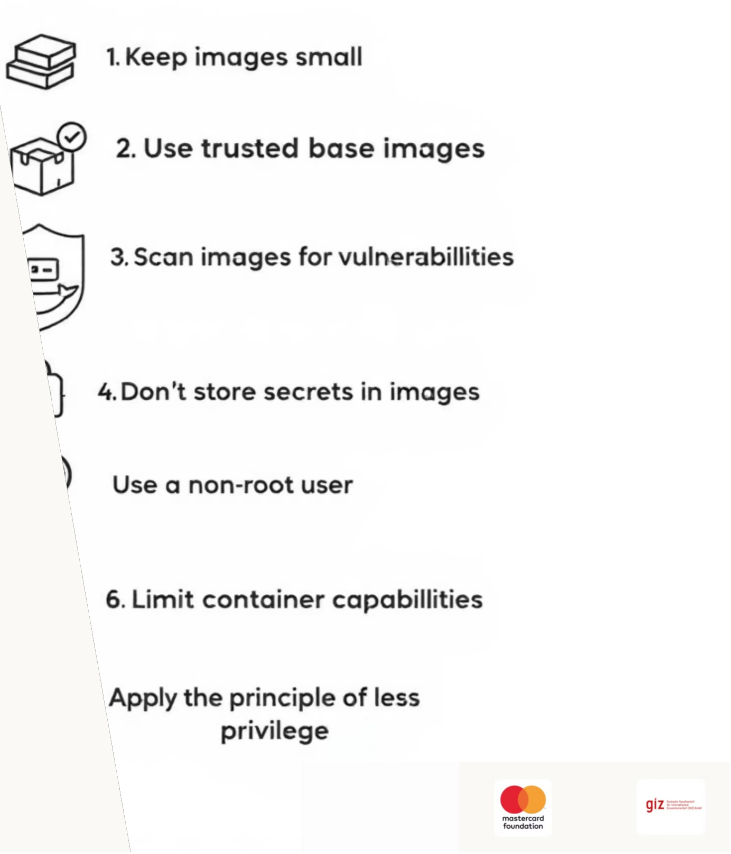
- Use Alpine or slim base images
- Multi-stage builds for production
- Remove unnecessary files and caches

Security First

- Never run containers as root
- Scan images for vulnerabilities
- Use secrets management properly

Optimise Layers

- Order Dockerfile commands strategically
- Combine RUN commands wisely
- Leverage build cache effectively



Your Docker Journey

Master the Fundamentals

Start with basic containers, understand images and volumes, practice essential commands until they're second nature.

Build Multi-Container Apps

Progress to Docker Compose, create realistic application stacks, learn how services interact and scale together.

Automate Everything

Integrate with CI/CD pipelines, automate builds and deployments, embrace infrastructure as code principles.

Production Excellence

Apply best practices, optimise for performance and security, monitor and maintain containerised systems at scale.

Continue Learning

Docker Projects Tutorial • Advanced Docker Concepts • Docker Production Patterns