

AN OPENSOURCE EBOOK

# INTRODUCTION TO



docker<sup>®</sup>

---

Bobby Iliev

# Table of Contents

|  |           |
|--|-----------|
| <b>About the book</b>                      | <b>10</b> |
| About the author                           | 11        |
| Sponsors                                   | 12        |
| Ebook PDF Generation Tool                  | 14        |
| Book Cover                                 | 15        |
| License                                    | 16        |
| <br>                                       |           |
| <b>Chapter 1: Introduction to Docker</b>   | <b>17</b> |
| What is Docker?                            | 18        |
| Why Use Docker?                            | 19        |
| Docker Architecture                        | 20        |
| Containers vs. Virtual Machines            | 21        |
| Basic Docker Workflow                      | 22        |
| Docker Components                          | 23        |
| Use Cases for Docker                       | 24        |
| Conclusion                                 | 25        |
| <br>                                       |           |
| <b>Chapter 2: Installing Docker</b>        | <b>26</b> |
| Docker Editions                            | 27        |
| Installing Docker on Linux                 | 28        |
| Installing Docker on macOS                 | 32        |
| Installing Docker on Windows               | 33        |
| Post-Installation Steps                    | 34        |
| Docker Desktop vs Docker Engine            | 35        |
| Troubleshooting Common Installation Issues | 36        |
| Updating Docker                            | 37        |

|  |           |
|--|-----------|
| Uninstalling Docker .....                              | 38        |
| Conclusion .....                                       | 39        |
| <b>Chapter 3: Working with Docker Containers .....</b> | <b>40</b> |
| Running Your First Container .....                     | 41        |
| Basic Docker Commands .....                            | 42        |
| Running Containers in Different Modes .....            | 44        |
| Port Mapping .....                                     | 45        |
| Working with Container Logs .....                      | 46        |
| Executing Commands in Running Containers .....         | 47        |
| Practical Example: Running an Apache Container .....   | 48        |
| Container Resource Management .....                    | 49        |
| Container Networking .....                             | 50        |
| Data Persistence with Volumes .....                    | 51        |
| Container Health Checks .....                          | 52        |
| Cleaning Up .....                                      | 53        |
| Conclusion .....                                       | 54        |
| <b>Chapter 4: What are Docker Images .....</b>         | <b>55</b> |
| Key Concepts .....                                     | 56        |
| Working with Docker Images .....                       | 57        |
| Building Custom Images .....                           | 59        |
| Image Tagging .....                                    | 60        |
| Pushing Images to Docker Hub .....                     | 61        |
| Image Layers and Caching .....                         | 62        |
| Multi-stage Builds .....                               | 63        |
| Image Scanning and Security .....                      | 64        |
| Best Practices for Working with Images .....           | 65        |
| Image Management and Cleanup .....                     | 66        |
| Conclusion .....                                       | 67        |

|   |            |
|---|------------|
| <b>Chapter 5: What is a Dockerfile</b>  | <b>68</b>  |
| Anatomy of a Dockerfile                 | 69         |
| Dockerfile Instructions                 | 70         |
| Best Practices for Writing Dockerfiles  | 74         |
| Advanced Dockerfile Concepts            | 76         |
| Conclusion                              | 77         |
| <br>                                    |            |
| <b>Chapter 6: Docker Networking</b>     | <b>78</b>  |
| Docker Network Drivers                  | 79         |
| Working with Docker Networks            | 80         |
| Deep Dive into Network Drivers          | 83         |
| Network Troubleshooting                 | 86         |
| Best Practices                          | 87         |
| Advanced Topics                         | 88         |
| Conclusion                              | 89         |
| <br>                                    |            |
| <b>Chapter 7: Docker Volumes</b>        | <b>90</b>  |
| Why Use Docker Volumes?                 | 91         |
| Types of Docker Volumes                 | 92         |
| Working with Docker Volumes             | 94         |
| Volume Drivers                          | 96         |
| Best Practices for Using Docker Volumes | 97         |
| Advanced Volume Concepts                | 98         |
| Troubleshooting Volume Issues           | 100        |
| Conclusion                              | 101        |
| <br>                                    |            |
| <b>Chapter 8: Docker Compose</b>        | <b>102</b> |
| Key Benefits of Docker Compose          | 103        |
| The docker-compose.yml File             | 104        |

|   |            |
|---|------------|
| Key Concepts in Docker Compose .....                            | 105        |
| Basic Docker Compose Commands .....                             | 106        |
| Advanced Docker Compose Features .....                          | 107        |
| Practical Examples .....  | 109        |
| Best Practices for Docker Compose .....                         | 115        |
| Scaling Services .....  | 116        |
| Networking in Docker Compose .....                              | 117        |
| Volumes in Docker Compose .....                                 | 118        |
| Conclusion .....  | 119        |
| <br><b>Chapter 9: Docker Security Best Practices .....</b>      | <b>120</b> |
| 1. Keep Docker Updated .....                                    | 121        |
| 2. Use Official Images .....                                    | 122        |
| 3. Scan Images for Vulnerabilities .....                        | 123        |
| 4. Limit Container Resources .....                              | 124        |
| 5. Use Non-Root Users .....                                     | 125        |
| 6. Use Secret Management .....                                  | 126        |
| 7. Enable Content Trust .....                                   | 127        |
| 8. Use Read-Only Containers .....                               | 128        |
| 9. Implement Network Segmentation .....                         | 129        |
| 10. Regular Security Audits .....                               | 130        |
| 11. Use Security-Enhanced Linux (SELinux) or AppArmor .....     | 131        |
| 12. Implement Logging and Monitoring .....                      | 132        |
| Conclusion .....  | 133        |
| <br><b>Chapter 10: Docker in Production: Orchestration with</b> |            |
| <b>Kubernetes .....</b>   | <b>134</b> |
| Key Kubernetes Concepts .....                                   | 135        |
| Setting Up a Kubernetes Cluster .....                           | 136        |
| Deploying a Docker Container to Kubernetes .....                | 137        |

|   |            |
|---|------------|
| Scaling in Kubernetes .....                                       | 139        |
| Rolling Updates .....   | 140        |
| Monitoring and Logging .....                                      | 141        |
| Kubernetes Dashboard .....  | 142        |
| Persistent Storage in Kubernetes .....                            | 143        |
| Kubernetes Networking .....                                       | 144        |
| Kubernetes Secrets .....  | 145        |
| Helm: The Kubernetes Package Manager .....                        | 146        |
| Best Practices for Kubernetes in Production .....                 | 147        |
| Conclusion .....  | 148        |
| <br><b>Chapter 11: Docker Performance Optimization .....</b>      | <b>149</b> |
| 1. Optimizing Docker Images .....                                 | 150        |
| 2. Container Resource Management .....                            | 152        |
| 3. Networking Optimization .....                                  | 153        |
| 4. Storage Optimization .....                                     | 154        |
| 5. Logging and Monitoring .....                                   | 155        |
| 6. Docker Daemon Optimization .....                               | 156        |
| 7. Application-Level Optimization .....                           | 157        |
| 8. Benchmarking and Profiling .....                               | 158        |
| 9. Orchestration-Level Optimization .....                         | 159        |
| Conclusion .....  | 160        |
| <br><b>Chapter 12: Docker Troubleshooting and Debugging .....</b> | <b>161</b> |
| 1. Container Lifecycle Issues .....                               | 162        |
| 2. Networking Issues .....  | 163        |
| 3. Storage and Volume Issues .....                                | 164        |
| 4. Resource Constraints .....                                     | 165        |
| 5. Image-related Issues .....                                     | 166        |
| 6. Docker Daemon Issues .....                                     | 167        |

|  |            |
|--|------------|
| 7. Debugging Techniques .....                                  | 168        |
| 8. Performance Debugging .....                                 | 169        |
| 9. Docker Compose Troubleshooting .....                        | 170        |
| Conclusion .....   | 171        |
| <b>Chapter 13: Advanced Docker Concepts and Features .....</b> | <b>172</b> |
| 1. Multi-stage Builds .....                                    | 173        |
| 2. Docker BuildKit .....                                       | 174        |
| 3. Custom Bridge Networks .....                                | 175        |
| 4. Docker Contexts .....                                       | 176        |
| 5. Docker Content Trust (DCT) .....                            | 177        |
| 6. Docker Secrets .....  | 178        |
| 7. Docker Health Checks .....                                  | 179        |
| 8. Docker Plugins .....  | 180        |
| 9. Docker Experimental Features .....                          | 181        |
| 10. Container Escape Protection .....                          | 182        |
| 11. Custom Dockerfile Instructions .....                       | 183        |
| 12. Docker Manifest .....                                      | 184        |
| 13. Docker Buildx .....  | 185        |
| 14. Docker Compose Profiles .....                              | 186        |
| Conclusion .....   | 187        |
| <b>Chapter 14: Docker in CI/CD Pipelines .....</b>             | <b>188</b> |
| 1. Docker in Continuous Integration .....                      | 189        |
| 2. Docker in Continuous Deployment .....                       | 190        |
| 3. Docker Compose in CI/CD .....                               | 192        |
| 4. Security Scanning .....                                     | 193        |
| 5. Performance Testing .....                                   | 194        |
| 6. Environment-Specific Configurations .....                   | 195        |
| 7. Caching in CI/CD .....                                      | 196        |

|   |     |
|---|-----|
| 8. Blue-Green Deployments with Docker ..... | 197 |
| 9. Monitoring and Logging in CI/CD .....    | 198 |
| Conclusion .....                            | 199 |

## **Chapter 15: Docker and Microservices Architecture ..... 200**

|                                      |     |
|--------------------------------------|-----|
| 1. Principles of Microservices ..... | 201 |
| 2. Dockerizing Microservices .....   | 202 |
| 3. Inter-service Communication ..... | 203 |
| 4. Service Discovery .....           | 205 |
| 5. API Gateway .....                 | 206 |
| 6. Data Management .....             | 207 |
| 7. Monitoring Microservices .....    | 208 |
| 8. Scaling Microservices .....       | 209 |
| 9. Testing Microservices .....       | 210 |
| 10. Deployment Strategies .....      | 211 |
| Conclusion .....                     | 212 |

## **Chapter 16: Docker for Data Science and Machine Learning .. 213**

|  |     |
|--|-----|
| 1. Setting Up a Data Science Environment .....           | 214 |
| 2. Managing Dependencies with Docker .....               | 215 |
| 3. GPU Support for Machine Learning .....                | 216 |
| 4. Distributed Training with Docker Swarm .....          | 217 |
| 5. MLOps with Docker .....                               | 218 |
| 6. Data Pipeline with Apache Airflow .....               | 219 |
| 7. Reproducible Research with Docker .....               | 220 |
| 8. Big Data Processing with Docker .....                 | 221 |
| 9. Automated Machine Learning (AutoML) with Docker ..... | 222 |
| 10. Hyperparameter Tuning at Scale .....                 | 223 |
| Conclusion .....   | 224 |



|  |                |
|--|----------------|
| <b>What is Docker Swarm mode .....</b> | <b>225</b>     |
| Docker Services .....                  | 226            |
| Building a Swarm .....                 | 227            |
| <br><b>Managing the cluster .....</b>  | <br><b>230</b> |
| Promote a worker to manager .....      | 232            |
| Using Services .....                   | 233            |
| Scaling a service .....                | 235            |
| Deleting a service .....               | 237            |
| Docker Swarm Knowledge Check .....     | 238            |
| <br><b>Conclusion .....</b>            | <br><b>239</b> |
| Other eBooks .....                     | 240            |

# About the book

- **This version was published on August 19 2024**

This is an introduction to Docker ebook that will help you learn the basics of Docker and how to start using containers for your SysOps, DevOps, and Dev projects. No matter if you are a DevOps/SysOps engineer, developer, or just a Linux enthusiast, you will most likely have to use Docker at some point in your career.

The guide is suitable for anyone working as a developer, system administrator, or a DevOps engineer and wants to learn the basics of Docker.

## About the author

My name is Bobby Iliev, and I have been working as a Linux DevOps Engineer since 2014. I am an avid Linux lover and supporter of the open-source movement philosophy. I am always doing that which I cannot do in order that I may learn how to do it, and I believe in sharing knowledge.

I think it's essential always to keep professional and surround yourself with good people, work hard, and be nice to everyone. You have to perform at a consistently higher level than others. That's the mark of a true professional.

For more information, please visit my blog at <https://bobbyiliev.com>, follow me on Twitter [@bobbyiliev\\_](#) and [YouTube](#).

## Sponsors

This book is made possible thanks to these fantastic companies!

### Materialize

The Streaming Database for Real-time Analytics.

Materialize is a reactive database that delivers incremental view updates. Materialize helps developers easily build with streaming data using standard SQL.

### DigitalOcean

DigitalOcean is a cloud services platform delivering the simplicity developers love and businesses trust to run production applications at scale.

It provides highly available, secure, and scalable compute, storage, and networking solutions that help developers build great software faster.

Founded in 2012 with offices in New York and Cambridge, MA, DigitalOcean offers transparent and affordable pricing, an elegant user interface, and one of the largest libraries of open source resources available.

For more information, please visit <https://www.digitalocean.com> or follow [@digitalocean](#) on Twitter.

If you are new to DigitalOcean, you can get a free \$100 credit and spin up your own servers via this referral link here:

[Free \\$200 Credit For DigitalOcean](#)

## **DevDojo**

The DevDojo is a resource to learn all things web development and web design. Learn on your lunch break or wake up and enjoy a cup of coffee with us to learn something new.

Join this developer community, and we can all learn together, build together, and grow together.

[Join DevDojo](#)

For more information, please visit <https://www.devdojo.com> or follow [@thedeveloper](#) on Twitter.

## **Ebook PDF Generation Tool**

This ebook was generated by [Ibis](#) developed by [Mohamed Said](#).

Ibis is a PHP tool that helps you write eBooks in markdown.

## **Book Cover**

The cover for this ebook was created with [Canva.com](https://www.canva.com).

If you ever need to create a graphic, poster, invitation, logo, presentation – or anything that looks good — give Canva a go.

# License

## MIT License

Copyright (c) 2020 Bobby Iliev

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



# **Chapter 1: Introduction to Docker**

# What is Docker?

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization technology. It allows developers to package applications and their dependencies into standardized units called containers, which can run consistently across different environments.

## Key Concepts:

1. **Containerization:** A lightweight form of virtualization that packages applications and their dependencies together.
2. **Docker Engine:** The runtime that allows you to build and run containers.
3. **Docker Image:** A read-only template used to create containers.
4. **Docker Container:** A runnable instance of a Docker image.
5. **Docker Hub:** A cloud-based registry for storing and sharing Docker images.

# Why Use Docker?

Docker offers numerous advantages for developers and operations teams:

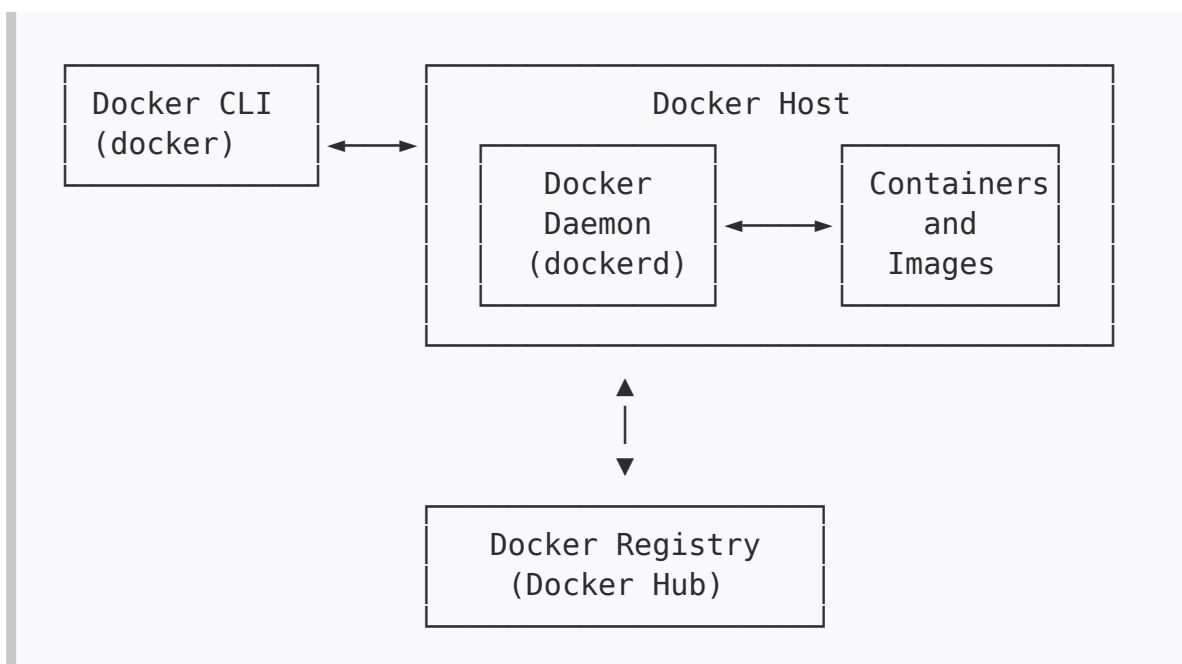
1. **Consistency:** Ensures applications run the same way in development, testing, and production environments.
2. **Isolation:** Containers are isolated from each other and the host system, improving security and reducing conflicts.
3. **Portability:** Containers can run on any system that supports Docker, regardless of the underlying infrastructure.
4. **Efficiency:** Containers share the host system's OS kernel, making them more lightweight than traditional virtual machines.
5. **Scalability:** Easy to scale applications horizontally by running multiple containers.
6. **Version Control:** Docker images can be versioned, allowing for easy rollbacks and updates.

# Docker Architecture

Docker uses a client-server architecture:

1. **Docker Client:** The primary way users interact with Docker through the command line interface (CLI).
2. **Docker Host:** The machine running the Docker daemon (dockerd).
3. **Docker Daemon:** Manages Docker objects like images, containers, networks, and volumes.
4. **Docker Registry:** Stores Docker images (e.g., Docker Hub).

Here's a simplified diagram of the Docker architecture:



## Containers vs. Virtual Machines

While both containers and virtual machines (VMs) are used for isolating applications, they differ in several key aspects:

| Aspect         | Containers                            | Virtual Machines            |
|----------------|---------------------------------------|-----------------------------|
| OS             | Share host OS kernel                  | Run full OS and kernel      |
| Resource Usage | Lightweight, minimal overhead         | Higher resource usage       |
| Boot Time      | Seconds                               | Minutes                     |
| Isolation      | Process-level isolation               | Full isolation              |
| Portability    | Highly portable across different OSes | Less portable, OS-dependent |
| Performance    | Near-native performance               | Slight performance overhead |
| Storage        | Typically smaller (MBs)               | Larger (GBs)                |

## Basic Docker Workflow

1. **Build:** Create a Dockerfile that defines your application and its dependencies.
2. **Ship:** Push your Docker image to a registry like Docker Hub.
3. **Run:** Pull the image and run it as a container on any Docker-enabled host.

Here's a simple example of this workflow:

```
# Build an image
docker build -t myapp:v1 .

# Ship the image to Docker Hub
docker push username/myapp:v1

# Run the container
docker run -d -p 8080:80 username/myapp:v1
```

## Docker Components

1. **Dockerfile**: A text file containing instructions to build a Docker image.
2. **Docker Compose**: A tool for defining and running multi-container Docker applications.
3. **Docker Swarm**: Docker's native clustering and orchestration solution.
4. **Docker Network**: Facilitates communication between Docker containers.
5. **Docker Volume**: Provides persistent storage for container data.

## Use Cases for Docker

1. **Microservices Architecture:** Deploy and scale individual services independently.
2. **Continuous Integration/Continuous Deployment (CI/CD):** Streamline development and deployment processes.
3. **Development Environments:** Create consistent development environments across teams.
4. **Application Isolation:** Run multiple versions of an application on the same host.
5. **Legacy Application Migration:** Containerize legacy applications for easier management and deployment.



## **Conclusion**

Docker has revolutionized how applications are developed, shipped, and run. By providing a standardized way to package and deploy applications, Docker addresses many of the challenges faced in modern software development and operations. As we progress through this book, we'll dive deeper into each aspect of Docker, providing you with the knowledge and skills to leverage this powerful technology effectively.

# Chapter 2: Installing Docker

Installing Docker is the first step in your journey with containerization. This chapter will guide you through the process of installing Docker on various operating systems, troubleshooting common issues, and verifying your installation.

# Docker Editions

Before we begin, it's important to understand the different Docker editions available:

1. **Docker Engine - Community:** Free, open-source Docker platform suitable for developers and small teams.
2. **Docker Engine - Enterprise:** Designed for enterprise development and IT teams building, running, and operating business-critical applications at scale.
3. **Docker Desktop:** An easy-to-install application for Mac or Windows environments that includes Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper.

For most users, Docker Engine - Community or Docker Desktop will be sufficient.

# Installing Docker on Linux

Docker runs natively on Linux, making it the ideal platform for Docker containers. There are two main methods to install Docker on Linux: using the convenience script or manual installation for specific distributions.

## Method 1: Using the Docker Installation Script (Recommended for Quick Setup)

Docker provides a convenient script that automatically detects your Linux distribution and installs Docker for you. This method is quick and works across many Linux distributions:

1. Run the following command to download and execute the Docker installation script:

```
wget -q0- https://get.docker.com | sh
```

2. Once the installation is complete, start the Docker service:

```
sudo systemctl start docker
```

3. Enable Docker to start on boot:

```
sudo systemctl enable docker
```

This method is ideal for quick setups and testing environments. However, for production environments, you might want to consider the manual installation method for more control over the process.

## Method 2: Manual Installation for Specific Distributions

For more control over the installation process or if you prefer to follow distribution-specific steps, you can manually install Docker. Here are instructions for popular Linux distributions:

Docker runs natively on Linux, making it the ideal platform for Docker containers. Here's how to install Docker on popular Linux distributions:

### Ubuntu

1. Update your package index:

```
sudo apt-get update
```

2. Install prerequisites:

```
sudo apt-get install apt-transport-https ca-certificates  
curl software-properties-common
```

3. Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |  
sudo apt-key add -
```

#### 4. Set up the stable repository:

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -  
cs) stable"
```

#### 5. Update the package index again:

```
sudo apt-get update
```

#### 6. Install Docker:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

## CentOS

#### 1. Install required packages:

```
sudo yum install -y yum-utils device-mapper-persistent-  
data lvm2
```

#### 2. Add Docker repository:

```
sudo yum-config-manager --add-repo  
https://download.docker.com/linux/centos/docker-ce.repo
```

This is a sample from "Introduction to Docker" by Bobby Iliev.

For more information, [Click here](#).